CodeWarrior Development Studio for Microcontrollers V10.x Targeting Manual

Revised: January 12, 2011

freescale™
semiconductor

Freescale, the Freescale logo, CodeWarrior, ColdFire, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Flexis and Processor Expert are trademarks of Freescale Semiconductor, Inc. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. All other product or service names are the property of their respective owners.

© 2009-2011 Freescale Semiconductor, Inc. All rights reserved.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

How to Contact Us

Corporate Headquarters	Freescale Semiconductor, Inc. 6501 William Cannon Drive West Austin, Texas 78735 U.S.A.
World Wide Web	`http://www.freescale.com/codewarrior`
Technical Support	`http://www.freescale.com/support`

Table of Contents

1 Introduction 13

Release Notes 13
About this Manual 13
Related Documentation. 15
    Additional Information Sources 15
    CodeWarrior Development Tools 22
    CodeWarrior Development Process 25

2 Working with Projects 29

Types of Projects. 29
    Bareboard Projects 29
    Linux Projects 30
New Bareboard Project Wizard 30
    Create an MCU Bareboard Project Page. 31
    Devices Page 32
    Connections Page for HCS08/ RS08 Derivative 34
    Connections Page for ColdFire V1 Derivative 35
    Connections Page for ColdFire V2-4 Derivative. 36
    Connections Page for MPC56xx Derivative 38
    Connections Page for Kinetis Derivative 40
    LSM/ DPM Configuration Page 41
    Add Files Page 42
    Languages Page. 44
    Languages and Build Tools Options 45
    C/C++ Options Page 47
    ColdFire Build Options Page for ColdFire V1 49
    ColdFire Build Options Page for ColdFire V2/3/4 51
    Rapid Application Development Page 53
    Processor Expert MCU Pin Variants and Configuration Page 54
New Linux/uClinux Application Project Wizard 55
    Create a Linux/uClinux Application Project Page 56
    Device used for Linux Application Debug Page. 57

Project Language and Output Page 58
Connections Page. 60
Application Debug Options Page 61
Creating Projects 62
Creating Bareboard Projects 62
Creating Linux/uClinux Application Project. 111
Building Projects 118
Manual-Build Mode 118
Auto-Build Mode. 120
Debugging Projects 120
Deleting Projects 122
Importing Classic CodeWarrior Projects. 123
Tutorials — Importing Connection-Specific Projects. 124
Tutorial A: Porting Classic HCS08 Project. 124
Tutorial B: Porting Classic RS08 Project 136
Tutorial C: Porting Classic ColdFire V1 Project. .149
Tutorial D: Porting Classic ColdFire V2/3/4 Project. 162
Tutorial E: Porting Classic Power Architecture Project 174

3 Build Properties for Bareboard Projects 191

Changing Build Properties 192
Restoring Build Properties 194
Defining C/C++ Build Settings and Behavior. 194
Define Build Settings. 194
Define Build Behavior. 197
Build Properties for HCS08 201
Messages 204
Host 206
General. 208
Disassembler 210
Linker. 215
Burner 231
HCS08 Compiler 236
HCS08 Assembler 282
Preprocessor. 297

Build Properties for RS08 299
- Messages 302
- Host 305
- General 307
- Disassembler 309
- Linker 314
- Burner 329
- RS08 Compiler 334
- RS08 Assembler 379
- Preprocessor 394

Build Properties for ColdFire 396
- ColdFire CPU 398
- Debugging 399
- Messages 401
- Librarian 402
- Burner 404
- ColdFire Linker 407
- ColdFire Compiler. 413
- ColdFire Assembler. 434
- ColdFire Preprocessor 440
- ColdFire Disassembler 443

Build Properties for Power Architecture. 446
- PowerPC CPU 448
- Debugging 452
- Messages 454
- Librarian 456
- PowerPC Linker 458
- PowerPC Compiler 470
- PowerPC Assembler 495
- PowerPC Disassembler 502
- PowerPC Preprocessor 505

Build Properties for ARM. 508
- ARM CPU 510
- Debugging 512
- Messages 513

Librarian 514
ARM Linker. 515
ARM Compiler 521
ARM Assembler 539
ARM Preprocessor. 545
ARM Disassembler 548

4 Working with Debugger 551
Initiating Debug Session 551
CodeWarrior Attach. 553
CodeWarrior Connect 555
CodeWarrior Download. 555
Customizing Launch Configuration 557
Main. 558
Arguments 583
Debugger 584
Source 605
Environment. 607
Common 608
Trace and Profile 610
Debugging Bare Board Software 613
Displaying Register Contents 614
Using Register Details Window. 617
Setting Watchpoints 625
Removing Watchpoints 628
Setting Breakpoints 629
Removing Breakpoints 632
Setting Stack Crawl Depth. 634
Changing Program Counter Value. 635
Viewing Memory 635
Hard Resetting 639
Debugging Externally Built Executable Files 639
Microcontrollers ELF Executable 640
Debug an Externally Built Microcontrollers Executable File 644

5 Multicore Debugging 657

Creating LSM/DPM Projects . . . 657
  Creating LSM Project . . . 658
  Creating DPM Project . . . 660
Debugging LSM/DPM Projects . . . 662
Debugging Multicore Projects . . . 665
  Targeting Core . . . 666
  Starting Debugging Session for Core . . . 666
  Debugging Specific Core . . . 667
  Multicore Operations. . . . 667
Editing Multicore Groups . . . 672
Editing System Types . . . 676
6 Memory Management Unit Viewer 679
MMU Viewer Pages . . . 679
  TLB . . . 680
  TLB Table . . . 683
MMU View . . . 683
  Opening MMU View . . . 684
7 Scripting 689
Tcl Support . . . 691
  Resolution of Conflicting Command Names . . . 691
  Execution of Script Files . . . 691
  Tcl Startup Script . . . 692
Command-Line Debugging Tasks . . . 693
Debugger Shell Command List . . . 694
  cmdwin::eppc::getcoreid . . . 694
  cmdwin::eppc::setMMRBaseAddr . . . 694
  cmdwin::eppc::setcoreid . . . 695
  gdi . . . 695
Microcontrollers-Specific HIWARE Commands . . . 696
  Command List . . . 696

8 Build Properties for Linux Project 709

Changing Build Properties 710
Restoring Build Properties 712
Build Properties for Linux/uClinux Project 712
  Architecture 714
  ColdFire uClinux Linker 715
  ColdFire uClinux Compiler. 724
  ColdFire uClinux Assembler. 737
  ColdFire uClinux Preprocessor 741
  ColdFire uClinux Disassembler. 743

9 Debugging Embedded Linux 747

Debugging Linux Application. 747
  Working With CodeWarrior TRK 747
  Viewing Multiple Processes and Threads 760
  Debugging Applications that use fork() and exec() System Calls. 760
Debugging Shared Library 771
Debugging U-Boot Loader 780
  Installing Board Support Package 780
  Configuring Linux Target Image Builder 781
  Debugging U-Boot in Flash. 790
  Debugging U-Boot in RAM 790
Debugging Linux Kernel. 791
  Checking CodeWarrior IDE Installation 791
  Setting Up Target Hardware 792
  Installing Board Support Package 794
  Configuring LTIB 794
  Configuring Kernel 796
  Creating CodeWarrior Project 800

10 Connections — HCS08 803

Changing Connection in IDE 803
P&E Full Chip Simulation 804
  Chip View 804

Module Options. 811
P&E HCS08 Multilink\Cyclone Pro\OSBDM 861
Connection Options. 861

11 Connections — RS08 889

Changing Connection in IDE 889
P&E Full Chip Simulation 890
Chip View 890
Module Options. 896
P&E RS08 Multilink/Cyclone PRO/OSBDM 926
Connection Options. 926

12 Connections — ColdFire V1 955

Changing Connection in IDE 955
P&E USB BDM Multilink/Cyclone Pro/OSBDM 956
Changing P&E Connection Settings 957
Connection Assistant. 962
Active Mode Menu Options 964
Advanced Programming/Debug Options 964
View Register Files Options 968
P&E USB BDM Multilink/Cyclone PRO/OSBDM Connection-Specific Options. 969
CCS. 983
USB TAP. 984
Ethernet 984
. . . . 985

13 Connections — ColdFire V2/3/4 987

Changing Connection in IDE 987
P&E ColdFire Multilink/Cyclone MAX/OSBDM 988
Changing P&E Connection Settings 989
Connection Assistant. 993
View Register Files Options 995
P&E ColdFire Multilink/Cyclone MAX/OSBDM Connection- Specific Options. 997

CCS 1012
USB TAP 1012
Ethernet 1013
. 1013

14 Connections — Power Architecture 1015

Changing Connection in IDE 1015
P&E PPCNEXUS Multilink/Cyclone MAX/OSJTAG 1016
Connection Options 1016
Changing P&E Connections Settings 1017
Connection Assistant 1021
Launch Configuration Settings 1021
Active Mode Menu Options 1023
Advanced Programming/Debug Options. 1023
View Register Files Options 1026
P&E PPCNexus Multilink/Cyclone MAX/OSJTAG Connection- Specific Options 1028

15 Connections — Kinetis 1039

Changing Connection in IDE 1039
P&E ARM Multilink/Cyclone MAX/OSJTAG. 1040
Connection Options 1040
Changing P&E Connections Settings 1041
Connection Assistant 1043
Launch Configuration Settings 1044
Active Mode Menu Options 1046
Advanced Programming/Debug Options. 1046
View Register Files Options 1050
P&E ARM Multilink/Cyclone MAX/OSJTAG Connection- Specific Options. 1051

16 Common Connection Features 1063

Working with Flash Programmer 1063
Use Pre-Defined Programming Task 1064
Create Flash Programmer Task 1066

Flash Programmer for MCF5441X 1071
Quick Access to Target Tasks 1072
Target Task Toolbar 1072
Fast Access to Flash Programmer 1073
Fast Access to Hardware Diagnostics 1073
Fast Access to Import/Export Memory 1074
Flash Programmer Tutorials 1074
Tutorial A: Import and Execute HCS08 Flash Task 1075
Tutorial B: Import and Execute ColdFire Flash Task 1077
Tutorial C: Create Erase Memory Task for HCS08 1079
Tutorial D: Create Erase Flash Memory Task for ColdFire 1084
Tutorial E: Create Download Program Task for ColdFire 1090
Tutorial F: Import and Execute Power Architecture Flash Task 1094
Tutorial G: Switching Between Lock-Step and Decoupled Parallel Modes . . . 1095
Tutorial H: Create and Execute Diagnostics Action Task 1099
Tutorial I: Dump Entire Flash 1103
Tutorial J: Change Protection of Sector 1104
Tutorial K: Fast Access to Target Tasks Editors 1106
Tutorial L: Programming with Simple Flash 1108
Tutorial M: Exporting Target Tasks 1111
Working with Hardware Diagnostics Window 1111
Manipulating Target Memory 1113
Creating Target Task to Import Memory 1113
Creating Target Task to Export Memory 1117
Fill Memory with Data Pattern 1121

Index 1127

1

Introduction

This manual explains how to use the CodeWarrior Development Studio for Microcontrollers V10.x product. This chapter presents an overview of the manual and introduces you to the CodeWarrior development tools and development process.

The topics in this chapter are:

- Release Notes — Lists new features, bug fixes, and incompatibilities
- About this Manual — Describes the contents of this manual
- Related Documentation — Describes supplementary CodeWarrior documentation, third-party documentation, and references to helpful code examples and Web sites
- CodeWarrior Development Tools — Describes the steps you take to write and debug programs with the CodeWarrior IDE
- CodeWarrior Development Process — Describes the CodeWarrior development process

Release Notes

Before using the CodeWarrior IDE, read the developer notes. These notes contain important information about last-minute changes, bug fixes, incompatible elements, or other topics that may not be included in this manual.

NOTE The release notes for specific components of the CodeWarrior IDE are located in the `Release_Notes` folder in the CodeWarrior installation directory.

If you are new to the CodeWarrior IDE, read this chapter and the Getting Started chapter. This chapter provides references to resources of interest for the new users; the Getting Started chapter helps you familiarize with the software features.

About this Manual

Each chapter of this manual describes a different area of software development. Table 1.1 lists the contents of this manual.

Table 1.1 Manual Contents

Chapter / Appendix	Description
Introduction	This chapter.
Working with Projects	Explains how to use the CodeWarrior tools to create and work with projects.
Build Properties for Bareboard Projects	Explains build properties for Microcontrollers bareboard project.
Working with Debugger	Explains how to use the CodeWarrior development tools to debug a program executing on the simulator or microcontroller.
Multicore Debugging	Explains how to define multiple, arbitrary groupings of cores and perform multicore operations.
Memory Management Unit Viewer	Explains each page of the Memory Mangement Unit viewer. You use these pages to configure MMU mapping. The tabbed interface of the **MMU Viewer** displays pages for configuration options and for generated code.
Scripting	Explains how CodeWarrior supports the command-line interface.
Build Properties for Linux Project	Explains build properties for Microcontrollers Linux project.
Debugging Embedded Linux	Explains how to use the CodeWarrior tools to debug embedded Linux programs on ColdFire hardware.
Connections — HCS08	Describes the features and settings of the connections that interface the CodeWarrior debugger with the HCS08-based bareboard target and allows it to debug program code on the target.
Connections — RS08	Describes the features and settings of the connections that interface the CodeWarrior debugger with the RS08-based bareboard target, and allow it to debug program code on the target.
Connections — ColdFire V1	Describes the features and settings of the connections that interface the CodeWarrior debugger with the ColdFire V1-based bare board target, and allow it to debug program code on the target.

Table 1.1 Manual Contents (*continued*)

Chapter / Appendix	Description
Connections — ColdFire V2/3/4	Describes the features and settings of the connections that interface the CodeWarrior debugger with the ColdFire V2/3/4-based bare board target, and allow it to debug program code on the target.
Connections — Power Architecture	Describes the features and settings of the connections that interface the CodeWarrior debugger with the Power Architecture bare board target.
Connections — Kinetis	Describes the features and settings of the connections that interface the CodeWarrior debugger with the Kinetis-based bare board target.arget.
Common Connection Features	Explains how to use the CodeWarrior hardware tools for board bring-up, test, and analysis. Also, explains how to manipulate target memory.

Related Documentation

This topic provides information about documentation related to the CodeWarrior IDE and Freescale Microcontrollers development.

- Additional Information Sources
- CodeWarrior Development Tools
- CodeWarrior Development Process

Additional Information Sources

- To view the online help for the CodeWarrior tools, first select **Help > Help Contents** from the IDE's menu bar. Next, select **Microcontrollers V10.x Targeting Manual** from the **Contents** list.
- For late-breaking information about new features, bug fixes, known problems, and incompatibilities, read the release notes in this folder:

  `<CWInstallDir>\MCU\Release_Notes`

  where *CWInstallDir* is the directory that CodeWarrior was installed into.
- For general information about the CodeWarrior IDE and debugger, refer to the Freescale Eclipse Extensions Guide in this folder:

  `<CWInstallDir>\MCU\Help\PDF`

NOTE The Freescale Eclipse Extensions Guide is a general guide that is also part of other CodeWarrior Eclipse-based products. Therefore, it describes the following features that are not available in Microcontrollers v10.1: Cache, Memory Management Unit (MMU) Configurator, and Multicores. Also, it shows figures that are not just specific to Microcontrollers v10.1, that is sometimes the screenshots are of other CodeWarrior products, like StarCore v10.x or Power Architectures v10.x.

Table 1.2 lists the additional documents you can refer to for more information about CodeWarrior for Microcontrollers 10.x. These documents are categorized according to the four different documentation types as Getting Started, User Guides, Application Notes, and Supporting Information.

Table 1.2 Related Documentation

Documentation Type	Document	Description	PDF Location
Getting Started	Microcontrollers V10.x Getting Started Guide	Contains information to get you started using the CodeWarrior Development Studio to develop software that targets the HCS08/ RS08, ColdFire and Power architectures	`<CWInstallDir>\MCU\Getting Started Guide for Microcontrollers.pdf`
	Microcontrollers V10.x Quick Start	Explains the steps to install Microcontrollers V10.x, and create and debug a project.	`<CWInstallDir>\MCU\Quick Start for Microcontrollers.pdf`
	CodeWarrior Project Importer Quick Start	Explains the steps to convert a classic CodeWarrior project into an Eclipse IDE project.	`<CWInstallDir>\MCU\CodeWarrior Project Importer Quick Start.pdf`
	Eclipse Quick Reference Card	Introduces you to the interface of CodeWarrior for Microcontrollers V10.x Eclipse-based IDE and provides a quick reference to the key bindings.	`<CWInstallDir>\MCU\Eclipse Quick Reference Card.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
Getting Started	HCS08 Profiling and Analysis for Microcontrollers V10.x Quick Start	Explains how to collect trace and critical code data after creating, building, and running a project on the HCS08 MC9S08QE128 target in the CodeWarrior for Microcontrollers version 10.x debugger.	`<CWInstallDir>\MCU\HCS08 Profiling and Analysis Quick Start for Microcontrollers.pdf`
	ColdFire Profiling and Analysis for Microcontrollers V10.x Quick Start	Explains how to collect trace and critical code data after creating, building, and running a project on the ColdFire V1 MCF51JM128 target in the CodeWarrior for Microcontrollers version 10.x debugger.	`<CWInstallDir>\MCU\ColdFire V1 Profiling and Analysis Quick Start for Microcontrollers.pdf`
	Ethernet TAP Quick Start	Explains how to set up the Ethernet TAP probe for Freescale microcontrollers and processors.	`<CWInstallDir>\MCU\Ethernet TAP Quick Start for Microcontrollers.pdf`
User Guide	Freescale Eclipse Extensions Guide	Explains extensions to the CodeWarrior Eclipse IDE across all CodeWarrior products.	`<CWInstallDir>\MCU\Help\PDF\Freescale Eclipse Extensions Guide.pdf`
	Microcontrollers V10.x Targeting Manual	Explains how to use CodeWarrior Development Studio for Microcontrollers V10.x	`<CWInstallDir>\MCU\Help\PDF\Targeting Microcontrollers.pdf`
	Microcontrollers V10.x HC08 Build Tools Reference Manual	Describes the compiler used for the Freescale 8-bit Microcontroller Unit (MCU) chip series.	`<CWInstallDir>\MCU\Help\PDF\MCU_HCS08_Compiler.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
User Guide	Microcontrollers V10.x RS08 Build Tools Reference Manual	Describes the ANSI-C/C++ Compiler used for the Freescale 8-bit Microcontroller Unit (MCU) chip series.	`<CWInstallDir>\MCU\Help\PDF\MCU_RS08_Compiler.pdf`
	Microcontrollers V10.x ColdFire Build Tools Reference Manual	Describes the compiler used for the Freescale 8-bit Microcontroller Unit (MCU) chip series	`<CWInstallDir>\MCU\Help\PDF\MCU_ColdFire_Compiler.pdf`
	Microcontrollers V10.x Power Architectures Processors Build Tools Reference Manual	Describes the compiler used for the Power Architectures Processors	`<CWInstallDir>\MCU\Help\PDF\MCU_Power-Architecture_Compiler.pdf`
	Microcontrollers V10.x Kinetis Build Tools Reference Manual	Describes the compiler used for the Freescale 32-bit Microcontroller Unit (MCU) chip series.	`<CWInstallDir>\MCU\Help\PDF\MCU_Kinetis_Compiler.pdf`
	Microcontrollers V10.x MISRA-C:2004 Compliance Exceptions for the HC(S)08, RS08, ColdFire, Kinetis and Power Architecture Libraries Reference Manual	Describes the MISRA-C:2004 compliance exceptions for the HC(S)08, RS08, ColdFire, Kinetis and Power Architecture libraries.	`<CWInstallDir>\MCU\Help\PDF\MISRA_C_2004 Compliance Exceptions.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
User Guide	CodeWarrior Development Tools EWL C Reference	Describes the contents of the Embedded Warrior Library for C. This document is available only in ColdFire Architecture.	`<CWInstallDir>\MCU\Help\PDF\EWL C Reference.pdf`
	CodeWarrior Development Tools EWL C++ Reference	Describes the contents of the Embedded Warrior Library for C++. This document is available only in ColdFire Architecture.	`<CWInstallDir>\MCU\Help\PDF\EWL C++ Reference.pdf`
	Microcontrollers V10.x HC(S)08/ RS08 Assembler Reference Manual	Explains how to use the HC(S)08/RS08 Macro Assembler	`<CWInstallDir>\MCU\Help\PDF\HCS08-RS08_Assembler_MCU_Eclipse.pdf`
	Microcontrollers V10.x ColdFire Assembler Reference Manual	Explains the assembly-language syntax and IDE settings for the ColdFire assemblers	`<CWInstallDir>\MCU\Help\PDF\ColdFire_Assembler_MCU_Eclipse.pdf`
	Microcontrollers V10.x Kinetis Assembler Manual	Explains the corresponding assembly-language syntax and IDE settings for these assemblers.	`<CWInstallDir>\MCU\Help\PDF\Kinetis_Assembler_MCU_Eclipse.pdf`
	Microcontrollers V10.x HC(S)08/ RS08 Build Tools Utilities Manual	Describes the following five CodeWarrior IDE utilities: SmartLinker, Burner, Libmaker, Decoder, and Maker.	`<CWInstallDir>\MCU\Help\PDF\Build_Tools_Utilities.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
User Guide	Microcontrollers V10.x Profiling and Analysis Users Guide	Explains the CodeWarrior Profiling and Analysis tools. These tools provide visibility into an application as it runs on the simulator and hardware. Developers can use these tools to understand how an application runs, as well as identify operational problems.	`<CWInstallDir>\MCU\Help\PDF\Profiling and Analysis Users Guide.pdf`
	USB TAP Users Guide	Explains the steps to develop and debug a number of processors and microcontroller using CodeWarrior USB TAP probe.	`<CWInstallDir>\MCU\Help\PDF\USB TAP Users Guide.pdf`
	Ethernet TAP Users Guide	Explains the steps to develop and debug a number of processors and microcontroller using CodeWarrior Ethernet TAP probe.	`<CWInstallDir>\MCU\Help\PDF\Ethernet TAP Users Guide.pdf`
	Open Source BDM-JM60 Users Guide	Describes an Open Source programming and debugging development tool designed to work with Freescale HCS08, RS08, Coldfire V1,V2, V3 and V4, and DSC56800E microcontrollers.	`<CWInstallDir>\MCU\Help\PDF\OSBDM-JM60_Users_Guide.pdf`
	Processor Expert Users Manual	Provides information about Processor Expert plug-in, which generates code from the Embedded Beans.	`<CWInstallDir>\MCU\Help\PDF\ProcessorExpertHelp.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
User Guide	Device Initialization Users Manual	Provides information about the user interface, creating a simple design, configuring a device, generating initialization code, and using it in your application.	`<CWInstallDir>\MCU\Help\PDF\DeviceInitHelp.pdf`
	Signal Processing Engine Auxiliary Processing Unit Programming Interface Manual	Helps programmers provide software that is compatible across the family of Power Architecture processors that use the signal processing engine auxiliary processing unit.	`<CWInstallDir>\MCU\Help\PDF\SPE Programming Interface Manual.pdf`
Application Note	AN3859 - Adding Device(s) to the CodeWarrior Flash Programmer for Microcontrollers V10.x	Explains how to use the Flash Tool Kit to support additional flash devices on the Flash Programmer for CodeWarrior Development Studio for Microcontrollers V10.x.	`<CWInstallDir>\MCU\Help\PDF\AN3859.pdf`
	AN3967 - How to Write Flash Programming Applets	Provides information on creating Flash configuration files for the Flash Programming interface.	`<CWInstallDir>\MCU\Help\PDF\AN3967.pdf`
	AN4095 - CodeWarrior Build Tools Options for Optimal Performance on the Power Architecture e200 Core	Provides information on CodeWarrior build tools options for optimal performance on the Power ARchitecture e200 Core	`<CWInstallDir>\MCU\Help\PDF\AN4095.pdf`

Table 1.2 Related Documentation (*continued*)

Documentation Type	Document	Description	PDF Location
Application Note	AN4104 - Converting Classic ColdFire Projects to Microcontrollers V10.x	Explains how to convert a ColdFire project created in CodeWarrior Development Studio for Microcontrollers V6.2 or CodeWarrior Development Studio for ColdFire Architectures V7.1 to CodeWarrior Development Studio for Microcontrollers V10.x	`<`*`CWInstallDir`*`>\MCU\Help\PDF\AN4104.pdf`
	AN4188 - RS08 Upper Memory Access	Provides the RS08 programmer with information about the RS08 Upper Memory Access for Microcontrollers V10.x.	`<`*`CWInstallDir`*`>\MCU\PDF\AN4188.pdf`
Supporting Information	Microcontrollers V10.x FAQ Guide	Lists most frequently asked or anticipated questions and answers to CodeWarrior Development Studio for Microcontrollers V10.x.	`<CWInstallDir>\MCU\Help\PDF\Microcontrollers_FAQ_Guide.pdf`

CodeWarrior Development Tools

Programming for Microcontroller processors is much like programming for any other CodeWarrior platform target. If you have not used CodeWarrior tools before, start by studying the Eclipse IDE, which is used to host the tools. Information on the Eclipse IDE is available in the next topic.

If you are an experienced CodeWarrior user, note that the CodeWarrior Microcontrollers V10.x environment uses the Eclipse IDE, whose user interface is substantially different from the classic CodeWarrior IDE.

NOTE For information on the interface differences, refer to the Freescale Eclipse Extensions Guide.

The following topics explain the CodeWarrior tools:

- Eclipse IDE

- Compiler
- Assembler
- Linker
- CodeWarrior Debugger
- CodeWarrior Profiling and Analysis

Eclipse IDE

The Eclipse IDE (Integrated Development Environment) is an open-source development environment that lets you develop and debug your software. It controls the project manager, the source code editor, the class browser, the compilers and linkers, and the debugger.

Those who are more familiar with command-line development tools may find the concept of a CodeWarrior project new. The Eclipse Workspace organizes all files related to your project. This lets you see your project at a glance and eases the organization and navigation between source code files.

The Eclipse IDE has an extensible architecture that uses plug-in compilers and linkers to target various operating systems and microprocessors. The IDE is hosted on Microsoft Windows and x86 Linux platforms. There are many development tools available for the IDE, including C, C++, and Java compilers for desktop and embedded processors

For more information about the Eclipse IDE, read the Eclipse documentation at:

`http://www.eclipse.org/documentation/.`

Compiler

The Microcontrollers C Compiler:

- conforms to the American National Standards Institute (ANSI) C standards.
- conforms to version 1 of the Microcontrollers Application Binary Interface (ABI) standards.
- supports a set of Digital Signal Processor (DSP) extensions.
- supports International Telecommunications Union (ITU)/European Telecommunications Standards Institute (ETSI) primitives for saturating arithmetic. Additional parameters are available for non-saturating arithmetic and double-precision arithmetic.
- allows standard C constructs for representing special addressing modes.
- supports a wide range of runtime libraries and runtime environments.
- optimizes for size, speed, or a combination of both, depending on options that you select.

The compiler can link all application modules before optimizing. By examining the entire linked application before optimizing, the compiler produces highly optimized code. The compiler performs many optimizations, such as:

- software pipelining
- instruction paralleling and scheduling
- data and address register allocation
- aggressive loop transformations, including automatic unrolling

NOTE For more information, refer to the Microcontrollers V10.x HC(S)08/RS08 Build Tools Utility Manual.

Assembler

The assembler translates assembly-language source code to machine-language object files or executable programs. You can provide the assembly-language source code, or the compiler can generate it.

For each assembly-language module in a build target, the Microcontrollers assembler can generate a file that lists the generated code side-by-side with the assembly-language source code.

NOTE For more information, refer to the *Microcontrollers Assembler User Guide*.

Linker

The Linker combines object files into a single executable file. You specify the link mappings of your program in a Linker Command File (LCF).

NOTE For more information, refer to the *Microcontrollers Linker User Guide.*

CodeWarrior Debugger

The CodeWarrior debugger lets you debug your software on both simulator and hardware targets.

NOTE The CodeWarrior debugger is also validated on the host machine running Microsoft® Vista® Business Edition.

CodeWarrior Profiling and Analysis

CodeWarrior Profiling and Analysis tools provide visibility into an application as it runs on the simulator and hardware. This visibility can help you understand how your application runs, as well as identify operational problems. The tools also provide user friendly data viewing features:

- Enables simultaneous step through trace data and the corresponding source and assembly code of that trace data
- Export source line information of the performance data generated by the simulator into an Excel file
- Export the trace and function data generated by simulator and target hardware into an Excel file
- Apply multi-level filters to isolate data
- Apply multi-level searches to find specific data
- Display results in an intuitive, user friendly manner in the trace, critical code, and performance views
- Show or hide columns and also reorder the columns
- Copy and paste a cell or a line of the trace, alu-agu and performance data generated by simulator and target hardware
- Control trace collection by using start and stop tracepoints to reduce the amount of unwanted trace events in the trace buffer making the trace data easier to read
- View the value of the DPU counters in form of graphs (pie charts and bar charts) while the application is in debug mode
- Display real time cycle count for simulated targets to allow quick monitoring of evolution of application in time

NOTE For more information, refer to the *Profiling and Analysis User Guide*.

CodeWarrior Development Process

While working with the CodeWarrior IDE, you proceed through various development stages, such as writing code, compiling and linking, and debugging. Refer to the Freescale Eclipse Extensions Guide for:

- Complete information on tasks such as editing, compiling, and linking
- Basic information on debugging

The difference between the CodeWarrior environment and traditional command-line environments is how the software helps you effectively manage your work.

Refer to the following topics for more information on working with an integrated environment like the Eclipse IDE.

- Project Files
- Editing Code
- Compiling
- Linking
- Debugging

Project Files

A CodeWarrior project is analogous to a set of make files, because a project can have multiple settings that are applied when building the project. For example, you can have one project that has both a debug version and a release version of your program. You can build one or the other, or both as you wish. The different settings used to launch your program within a single project are called *launch configurations*.

The IDE uses the **CodeWarrior Projects** view to list all the files in a project. The files listed in the **CodeWarrior Projects** view include source code files and libraries.

You can add or remove files easily. You can also assign files to one or more different build configurations within the project, therefore files common to multiple build configurations can be managed simply.

The IDE automatically manages all the interdependencies between files and tracks which files have changed since the last build. This speeds the build process because the IDE only compiles those files that have changed since the last build.

In addition, the IDE stores the settings for compiler and linker options for each build configuration. You can modify these settings using the IDE, or with `#pragma` statements in your code.

Editing Code

The Eclipse IDE has an integral text editor designed for programmers. It handles text files in MS-DOS/Windows® and UNIX® formats.

To edit a source code file or any other editable file in a project, double-click the filename in the **CodeWarrior Projects** view to open the file.

The navigational features of the editor window lets you switch between related files, locate a particular function, mark a location within a file, or go to a specific line of code.

Compiling

To compile a source code file, ensure that the file is a part of the current launch configuration. If the file is in the configuration, select it in the **CodeWarrior Projects** view and select **Project > Build Project** from the IDE menu bar.

To automatically compile all the files in the current launch configuration after you modify them, select **Project > Build Automatically** from the IDE menu bar.

Linking

Select **Project > Build Project** from the IDE menu bar to link object code into a final binary file. The **Build Project** command makes the active project up-to-date and then links the resulting object code into a final output file.

You control the linker through the IDE. There is no need to specify a list of object files. The Workspace tracks all the object files automatically.

You can modify the build configuration settings to select the name of the final output file.

Debugging

Select **Run** > **Debug Configurations** from the IDE menu bar to debug your project. This command downloads the current project’s executable to the target board and starts a debug session.

NOTE You must have previously entered debugger settings for the launch configuration by choosing **Run > Debug Configurations**. The IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

You can now use the debugger to step through the program's code, view and change the value of variables, set breakpoints, and much more. Refer to the *Freescale Eclipse Extensions Guide* and the Working with Debugger chapter of this manual for instructions that explain how to use the debugger.

2

Working with Projects

This chapter explains how to use the CodeWarrior tools to create and work with projects.

The topics in this chapter are:

- Types of Projects
- New Bareboard Project Wizard
- New Linux/uClinux Application Project Wizard
- Creating Projects
- Building Projects
- Debugging Projects
- Deleting Projects
- Importing Classic CodeWarrior Projects
- Tutorials — Importing Connection-Specific Projects

Types of Projects

CodeWarrior projects organize files and various compiler, linker, and debugger settings associated with the applications or libraries you develop. You use Microcontrollers New Project Wizard to create new projects that group these files and settings into build and launch configurations. This section describes the different types of projects you can create:

- Bareboard Projects
- Linux Projects

Bareboard Projects

With CodeWarrior Development Studio for Microcontrollers, you can create a variety of projects that create ELF executable binary files that run directly on a given target simulator or board, without a Linux operating system. The type of project you create is based on selections you make in the **New Bareboard Project** wizard.

Linux Projects

You can create projects that generate Linux ELF executable binary files for applications. The project's type is determined by the options you choose in the **New Linux/ uClinus Application Project** wizard.

New Bareboard Project Wizard

The Microcontrollers **New Bareboard Projec**t wizard presents a series of pages that prompt you prompt you for the features and settings to be used when making your program. For example, the devices options lets you select the derivative or board you would like to use. This wizard also helps you specify other settings, such as whether the program executes on an emulator or simulator rather than actual hardware, and the characteristics of the connection that communicates with a hardware target.

This topic describes the various pages that the wizard displays as it assists you in creating a bareboard project. The pages of the wizard can differ based on the project type or execution target.

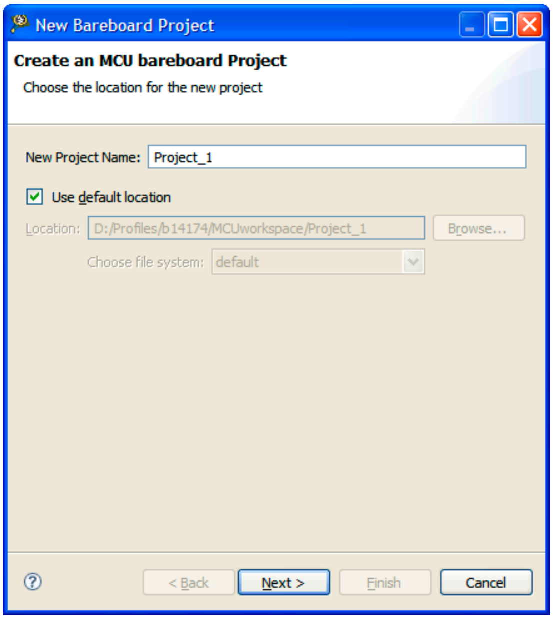
The pages of the **New Bareboard Project** wizard are:

- Create an MCU Bareboard Project Page
- Devices Page
- Connections Page for HCS08/ RS08 Derivative
- Connections Page for ColdFire V1 Derivative
- Connections Page for ColdFire V2-4 Derivative
- Connections Page for MPC56xx Derivative
- Connections Page for Kinetis Derivative
- LSM/ DPM Configuration Page
- Add Files Page
- Languages Page
- Languages and Build Tools Options
- C/C++ Options Page
- ColdFire Build Options Page for ColdFire V1
- ColdFire Build Options Page for ColdFire V2/3/4
- Rapid Application Development Page
- Processor Expert MCU Pin Variants and Configuration Page

Create an MCU Bareboard Project Page

Use this page to specify the project name and the directory where the project files are located.

Figure 2.1 Create an MCU Bareboard Project Page



Create an MCU bareboard Project

Choose the location for the new project

New Project Name: Project_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/Project_1

Choose file system: default

Table 2.1 describes the purpose of the various options.

Table 2.1 Create an MCU Bareboard Project Page Settings

Option	Description
New Project Name	Enter the name for the new project in this text box.
Use default location	Stores the files required to build the program in the Workbench's current workspace directory. The project files are located in the directory you specify. Use the **Location** option to select the directory.

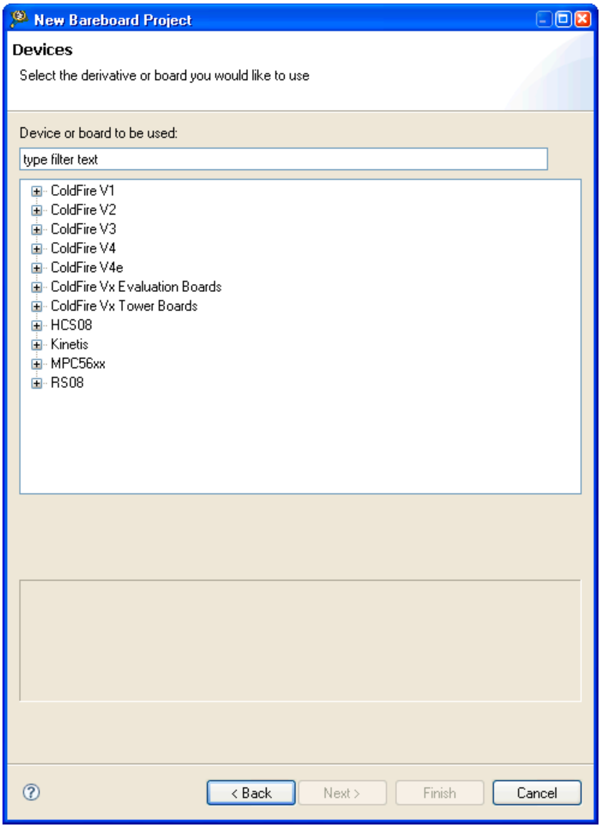
Table 2.1 Create an MCU Bareboard Project Page Settings (*continued*)

Option	Description
Location	Specifies the directory that contains the project files. Click **Browse** to navigate to the desired directory. This option is available only when **Use default location** is clear.
Choose file system	Specifies the file system to use. However, this option is available only when **Use default location** is clear. You can select either the default file system or a Remote System Explorer (RSE). For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

Devices Page

Use this page to select the derivative or board you would like to use.

Figure 2.2 Devices Page



Devices

Select the derivative or board you would like to use

Device or board to be used:

type filter text

- ColdFire V1
- ColdFire V2
- ColdFire V3
- ColdFire V4
- ColdFire V4e
- ColdFire Vx Evaluation Boards
- ColdFire Vx Tower Boards
- HCS08
- Kinetis
- MPC56xx
- RS08

NOTE The pages of the wizard change depending on the selected derivative or board. If an **MPC56xx** derivative or board is selected then the **LSM/ DPM configuration** page (Figure 2.8) appears in the wizard. Additionally, the **Languages** page is replaced by the **Languages and Build Tools Options** page. If a **ColdFire** derivative or board is selected then the **ColdFire Build Options** page (Figure 2.13 and Figure 2.14) appears in the wizard.

NOTE The MPC56xxL (Leopard) derivative currently supported in CodeWarrior MPC55xx/MPC56xx v2.x is MPC5643L.

NOTE The MPC56xxP (Pictus) derivatives supported in CodeWarrior MPC55xx/ MPC56xx v2.x are: MPC5601P *M07N, MPC5602P *M07N, MPC5603P

*M07N, MPC5604P *M07N, MPC5601P *M26V, MPC5602P *M26V, MPC5603P *M26V, and MPC5604P *M26V. However, not all derivatives are supported in this build.

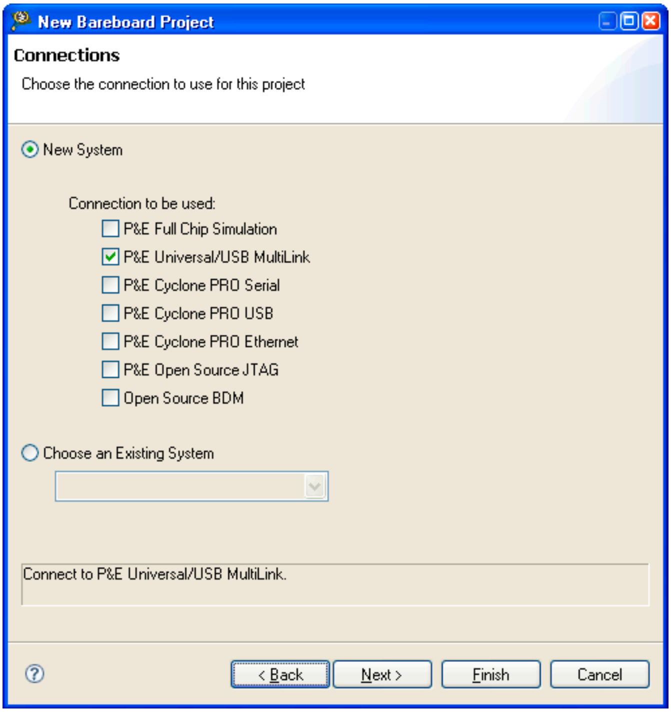
NOTE The MPC567xK (Komodo) derivatives supported in CW MPC55xx/MPC56xx v2.x are: MPC5671K, MPC5672K, MPC5673K, MPC5674K, and MPC5675K. However, not all derivatives are supported in this build.

Connections Page for HCS08/ RS08 Derivative

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

If an HCS08 derivative or board is selected in the **Devices** page, the **Connections** page appears as in Figure 2.3.

Figure 2.3 Connections Page — HCS08/RS08 Derivative



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☑ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ P&E Open Source JTAG
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

Table 2.2 explains the connections available on the **Connections** page (for HCS08/RS08).

Table 2.2 Connections Page Settings for HCS08/RS08 Derivative

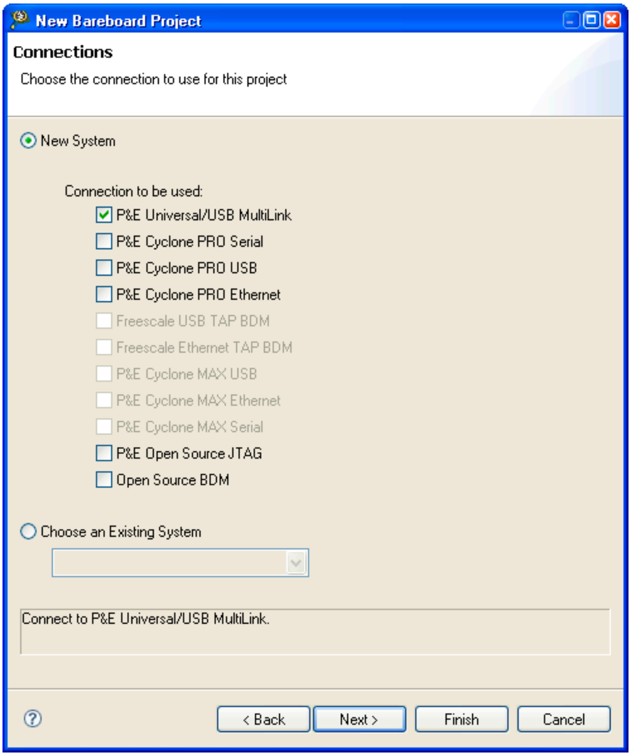
Option	Description
P&E Full Chip Simulation	Connect to P&E Full Chip Simulation for simulation of all on-chip peripherals
P&E Universal/USB Multilink	Connect to PE Universal/USB Multilink
P&E Cyclone PRO Serial	Connect to PE Cyclone PRO Serial
P&E Cyclone PRO USB	Connect to PE Cyclone PRO USB
P&E Cyclone PRO Ethernet	Connect to PE Cyclone PRO Ethernet
P&E Open Source JTAG	Connect to the PE Open Source JTAG
Open source BDM	Connect to the USB-based Freescale Open Source BDM Cable
Choose an Existing System	Select an existing remote system configuration. For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

Connections Page for ColdFire V1 Derivative

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

If a ColdFire V1 derivative or board is selected in the **Devices** page, the **Connections** page appears as in Figure 2.4.

Figure 2.4 Connections Page — ColdFire V1 Derivative



Connections

Choose the connection to use for this project

● New System

Connection to be used:

☑ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ P&E Open Source JTAG
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

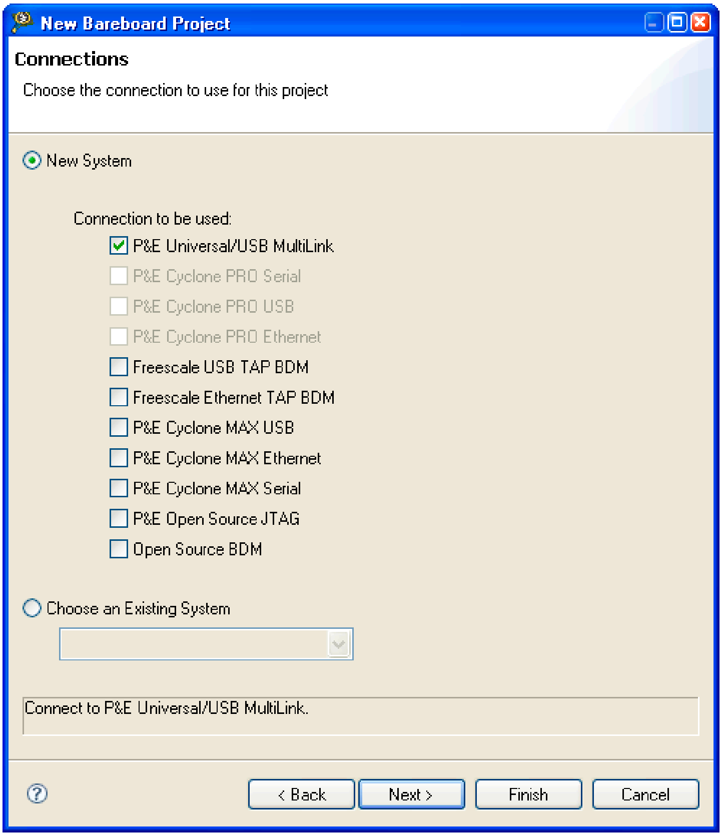
Table 2.3 explains the connections available on the **Connections** page for (ColdFire derivatives).

Connections Page for ColdFire V2-4 Derivative

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

If a ColdFire V2-4 derivative or board is selected in the **Devices** page, the **Connections** page appears as in Figure 2.5.

Figure 2.5 Connections Page — ColdFire V2-4 Derivative



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☑ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☐ P&E Open Source JTAG
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

Table 2.3 explains the connections available on the **Connections** page.

Table 2.3 Connections Page Settings for ColdFire Derivative

Option	Description
P&E USB Universal/USB Multilink	Connect to PE Universal/USB Multilink
P&E Cyclone PRO Serial	Connect to PE Cyclone PRO Serial
P&E Cyclone PRO USB	Connect to PE Cyclone PRO USB
P&E Cyclone PRO Ethernet	Connect to PE Cyclone PRO Ethernet
Freescale USB TAP BDM	Connect to Freescale USB TAP BDM

Table 2.3 Connections Page Settings for ColdFire Derivative (*continued*)

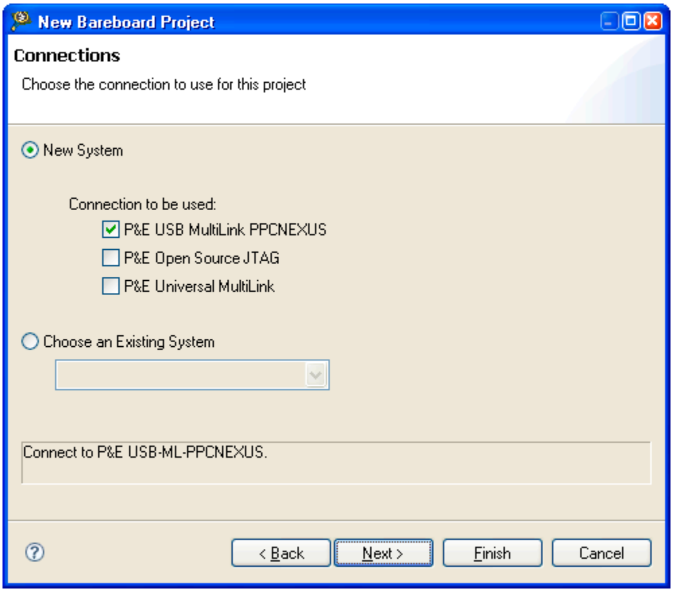
Option	Description
Freescale Ethernet TAP BDM	Connect to Freescale Ethernet TAP BDM
P&E Cyclone Max USB	Connect to PE Cyclone Max through host USB port
P&E Cyclone Max Ethernet	Connect to PE Cyclone Max through Ethernet
P&E Cyclone Max Serial	Connect to PE Cyclone Max through host serial port
P&E Open Source JTAG	Connect to PE Open Source JTAG
Open Source BDM	Connect to the USB-based Freescale Open Source BDM Cable
Choose an Existing System	Select an existing remote system configuration.

Connections Page for MPC56xx Derivative

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

If an MPC56xx derivative or board is selected in the **Devices** page, the **Connections** page appears as in Figure 2.6.

Figure 2.6 Connections Page — MPC56xx Derivative



● New System

Connection to be used:

☑ P&E USB MultiLink PPCNEXUS
☐ P&E Open Source JTAG
☐ P&E Universal MultiLink

○ Choose an Existing System

Connect to P&E USB-ML-PPCNEXUS.

Table 2.4 explains the connections available on the **Connections** page.

Table 2.4 Connections Page Settings for MPC56xx Derivative

Option	Description
P&E USB Multilink PPCNEXUS	Connect to P&E USB Multilink PPCNEXUS.
P&E Open Source JTAG	Connect to P&E Open Source JTAG.
P&E Universal Multilink	Connect to P&E Universal Multilink.
Choose an Existing System	Select an existing remote system configuration. For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

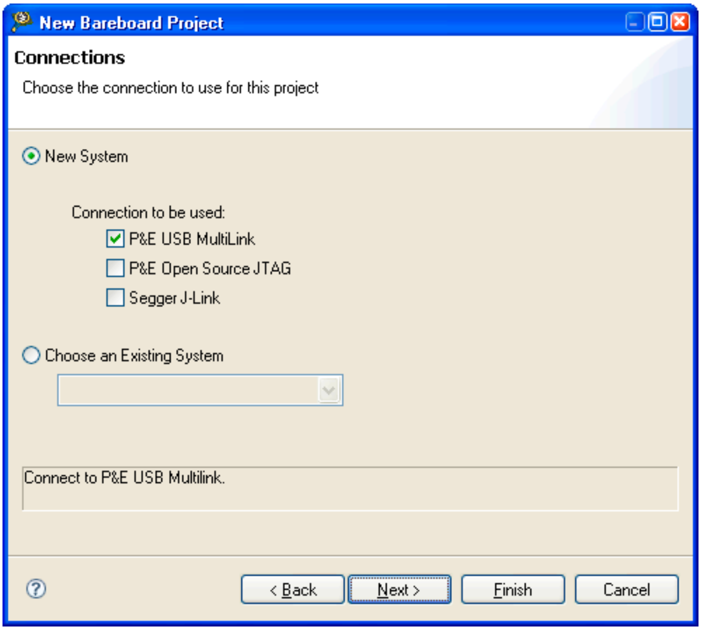
NOTE The CodeWarrior debugger supports NEXUS ISTO IEEE 5001-2003 and NEXUS ISTO IEEE 5001-2010 for MPC56xx families.

Connections Page for Kinetis Derivative

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

If an Kinetis derivative or board is selected in the **Devices** page, the **Connections** page appears as in Figure 2.7.

Figure 2.7 Connections Page — Kinetis Derivative


New Bareboard Project
Connections
Choose the connection to use for this project
● New System
Connection to be used:
☑ P&E USB MultiLink
☐ P&E Open Source JTAG
☐ Segger J-Link
○ Choose an Existing System
< Back
Next >
Finish
Cancel

Connect to P&E USB Multilink.

Table 2.5 explains the connections available on the **Connections** page.

Table 2.5 Connections Page Settings for Kinetis Derivative

Option	Description
P&E USB Multilink	Connect to P&E USB Multilink.
P&E Open Source JTAG	Connect to P&E Open Source JTAG.

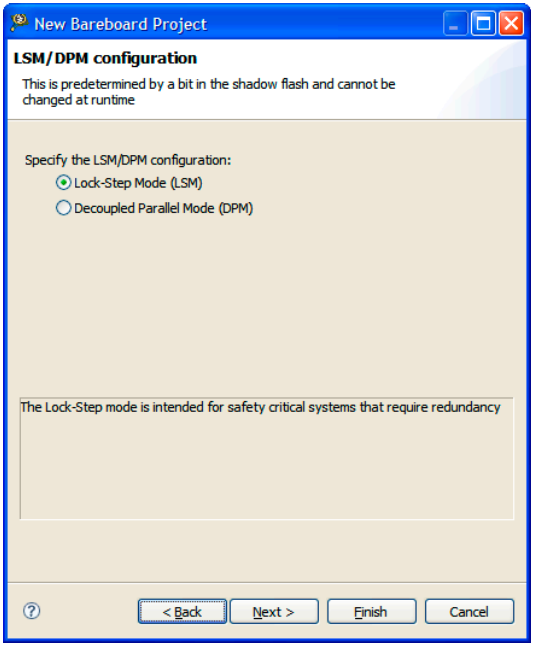
Table 2.5 Connections Page Settings for Kinetis Derivative (*continued*)

Option	Description
Segger J-Link	Connect to Segger J-Link.
Choose an Existing System	Select an existing remote system configuration. For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

LSM/ DPM Configuration Page

Use this page to select the **Lock-Step Mode (LSM)** and **Decoupled Parallel Mode (DPM)** configuration. This is predetermined by a bit in the shadow flash and cannot be changed at runtime.

Figure 2.8 LSM/ DPM Configuration Page



LSM/DPM configuration

This is predetermined by a bit in the shadow flash and cannot be changed at runtime

Specify the LSM/DPM configuration:

● Lock-Step Mode (LSM)

○ Decoupled Parallel Mode (DPM)

The Lock-Step mode is intended for safety critical systems that require redundancy

Table 2.6 explains the connections available on the **LSM/ DPM Configuration** page.

Table 2.6 LSM/ DPM Configuration Page Settings

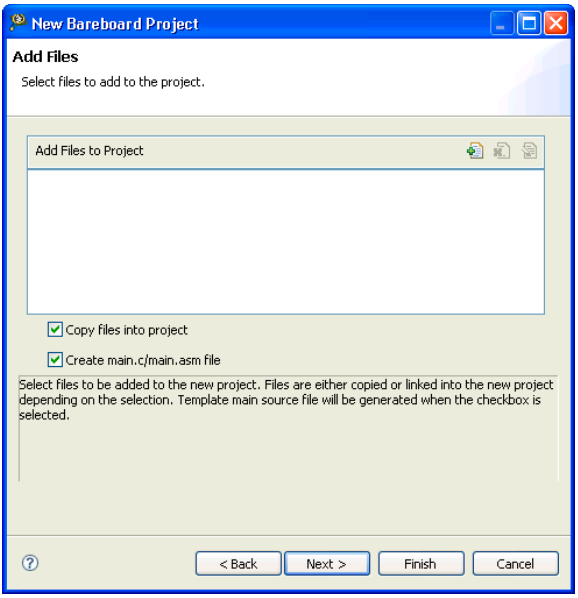
Option	Description
Lock-Step Mode (LSM)	Intended for safety critical systems that require redundancy.
Decoupled Parallel Mode (DPM)	Intended to increase performances that can be estimated in first approximation as about 1.6x the performance of the LS mode.

NOTE Many devices in the 55xx/56xx family are multicore devices (multiple e200 and eTPU cores). The debugger shall provide multicore debugging for the 55xx/56xx devices having multicore built in. This extends to both lock-step mode (LS mode or LSM) as well to Decoupled Parallel Mode (DP mode or DPM).

Add Files Page

Use this page to select files that you want to add to the project. Depending on the selection, you can either copy or link the files in the new project.

Figure 2.9 Add Files Page



Add Files

Select files to add to the project.

Add Files to Project

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

Table 2.7 describes the purpose of the various options.

Table 2.7 Add Files Page Settings

Option	Description
➕	**Add —** Click to open the **Add file path** dialog box and specify the location of the file you want to add.
✖	**Delete —** Click to delete the selected file path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit file path** dialog box and update the selected path.

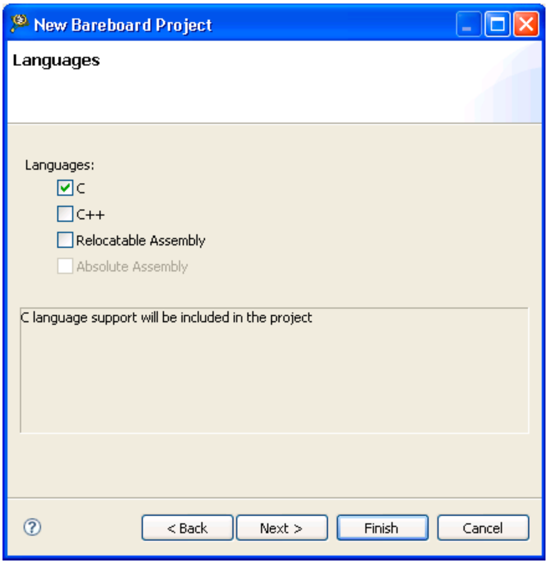
Table 2.7 Add Files Page Settings (*continued*) (*continued*)

Option	Description
Copy files into project	Check to add any existing files to your project. Clear if no files are to be added to your project.
Create main.c/main.asm file	Check to enable the IDE to create template files, including a Sources folder, in the project directory, along with some sample source-code files.

Languages Page

Use this page to select the programming language that you want to use when writing the program's source code. You can make multiple selections, creating the code in multiple formats.

Figure 2.10 Languages Page



Languages:
☑ C
☐ C++
☐ Relocatable Assembly
☐ Absolute Assembly

C language support will be included in the project

NOTE This page will not appear for ColdFire and MPC5xx derivatives.

Table 2.8 explains the options available on this page.

Table 2.8 Languages Page Settings

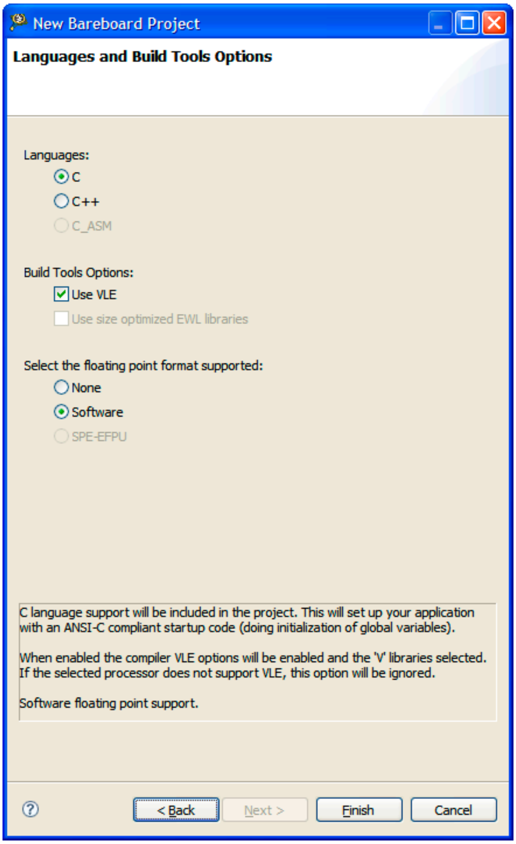
Group / Option	Description
C	Checking the **C** checkbox sets up your application with ANSI C-compliant startup code, and initializes global variables.
C++	Checking the **C++** checkbox sets up your application with ANSI C++ startup code, and performs global class object initialization.
Relocatable Assembly	Checking the **Relocatable** checkbox enables you to split up the application into multiple assembly source files. The source files are linked together using the linker.
Absolute Assembly	Checking the **Absolute Assembly** checkbox enables you to use only one single assembly source file with absolute assembly. There is no support for relocatable assembly or linker.

NOTE The option you select also sets up default compiler/linker options for the toolchain. For example, if you plan to use the C language in your source code files, check the **C** checkbox. If you plan to write the program using C++, check the **C++** checkbox.

Languages and Build Tools Options

Use this page to select the programming language, build tools options, and float point format support that you want to use when writing the program’s source code.

Figure 2.11 Languages and Build Tools Options Page



Languages:
● C
○ C++
○ C_ASM

Build Tools Options:
☑ Use VLE
☐ Use size optimized EWL libraries

Select the floating point format supported:
○ None
● Software
○ SPE-EFPU

C language support will be included in the project. This will set up your application with an ANSI-C compliant startup code (doing initialization of global variables).

When enabled the compiler VLE options will be enabled and the 'V' libraries selected. If the selected processor does not support VLE, this option will be ignored.

Software floating point support.

Table 2.9 explains the options available on this page.

Table 2.9 Languages and Build Tools Options Page Settings

Group / Option	Description
Languages	
C	Checking the **C** checkbox sets up your application with ANSI C-compliant startup code, and initializes global variables.
C++	Checking the **C++** checkbox sets up your application with ANSI C++ startup code, and performs global class object initialization.

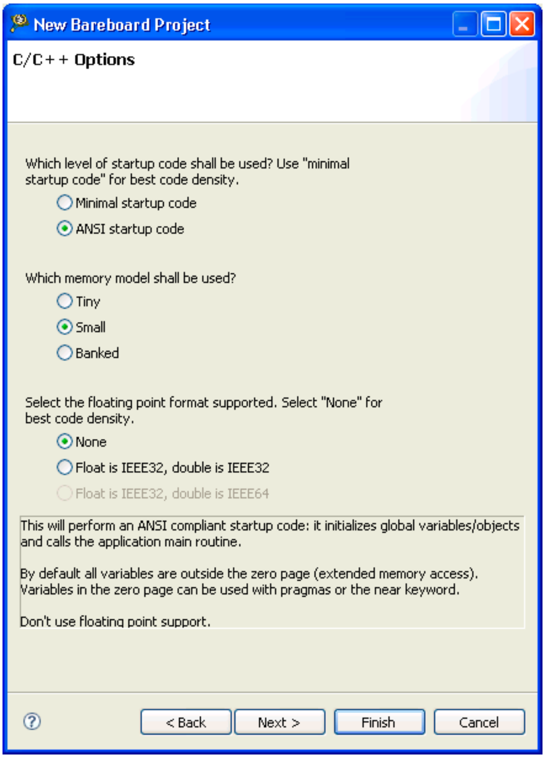
Table 2.9 Languages and Build Tools Options Page Settings (*continued*)

Group / Option	Description
C_ASM	
Build Tools Options	
Use VLE	Check to enable the compiler VLE options and select the V libraries. This checkbox is ignored, if the selected processor does not support VLE.
Use size optimized EWL libraries	
Select the floating point format support	
None	Select if you do not want to include floating point support; gives best code density.
Software	Select if you want to include software floating point support.
SPE-EPFU	

C/C++ Options Page

Use this page to select the level of startup code you want to produce, the memory model, and the appropriate floating point format support.

Figure 2.12 C/C++ Options Page



C/C++ Options

Which level of startup code shall be used? Use "minimal startup code" for best code density.

- ○ Minimal startup code
- ● ANSI startup code

Which memory model shall be used?

- ○ Tiny
- ● Small
- ○ Banked

Select the floating point format supported. Select "None" for best code density.

- ● None
- ○ Float is IEEE32, double is IEEE32
- ○ Float is IEEE32, double is IEEE64

This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.

By default all variables are outside the zero page (extended memory access). Variables in the zero page can be used with pragmas or the near keyword.

Don't use floating point support.

Table 2.10 explains the options available on this page.

Table 2.10 C/C++ Options Page Settings

Option	Description
Minimal startup code	Produces the best code density. The startup code initializes the stack pointer and calls the main function. No initialization of global variables is done, giving you the best speed/ code density and a fast startup time. The application code must address variable initialization. ANSI requires variable initialization and therefore this option is not ANSI compliant.
ANSI startup code	Initializes global variables/objects and calls the application main routine.

Table 2.10 C/C++ Options Page Settings (*continued*)

Option	Description
Tiny	Assumes that data pointers have 8-bit addresses unless explicitly specified with the keyword __far.
Small	Use the Small memory model if both the code and the data fit into the 64-kilobyte address space. By default, all variables and functions are accessed with 16-bit addresses. The compiler supports banked functions or paged variables in this memory model, but all accesses must be explicitly handled.
Banked	Banked memory model uses banked function calls by default, but the default data access is still 16-bit. Because the overhead of the far function call is not very large, this memory model suits all applications with more than 64-kilobytes of code. Data paging can be used, however all far objects and pointers to them must be specially declared.
None	Select for the best code intensity.
Float is IEEE32, double is IEEE32	All float and double variables are 32-bit IEEE32.
Float is IEEE32, double is IEEE64	Float variables are 32-bit IEEE32. Double variables are 64-bit IEEE64.

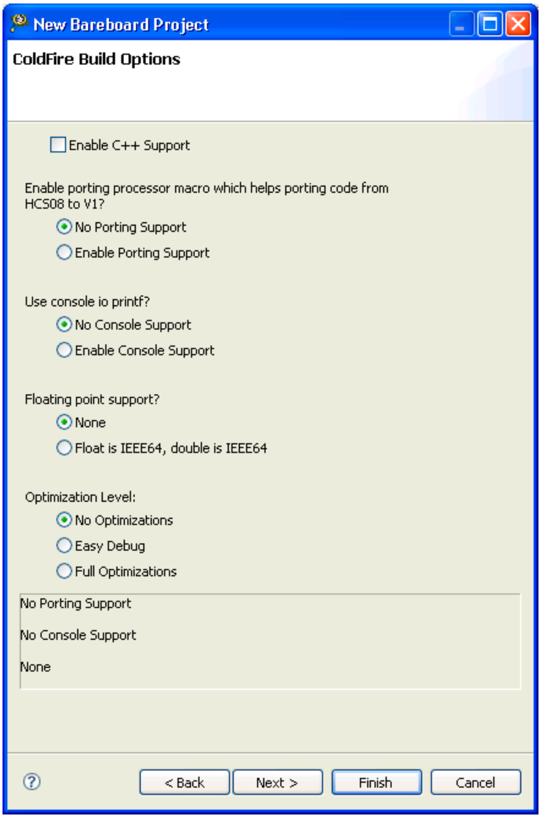
NOTE This page will not appear for ColdFire and MPC5xx derivatives.

ColdFire Build Options Page for ColdFire V1

Use this page to enable C++, porting processor macro, console, floating point support, and optimization level for ColdFire V1 derivatives.

NOTE This page will appear only if a ColdFire V1 derivative or board is selected in the **Devices** page (Figure 2.2). To enable the **Absolute Assembly** checkbox, you must uncheck the C and C++ options.

Figure 2.13 ColdFire Build Options Page — ColdFire V1



☐ Enable C++ Support

Enable porting processor macro which helps porting code from HCS08 to V1?

● No Porting Support
○ Enable Porting Support

Use console io printf?

● No Console Support
○ Enable Console Support

Floating point support?

● None
○ Float is IEEE64, double is IEEE64

Optimization Level:

● No Optimizations
○ Easy Debug
○ Full Optimizations

No Porting Support
No Console Support
None

Table 2.11 explains the options available on this page.

Table 2.11 ColdFire V1 Build Options Page Settings

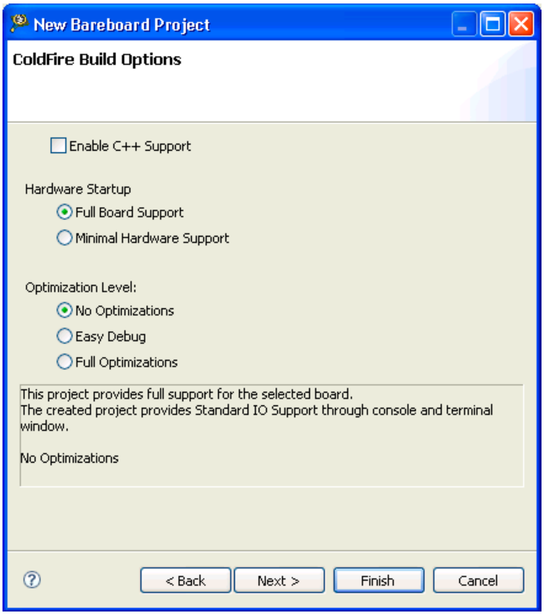
Option	Description
Enable C++ Support	Check to enable C++ support
No Porting Support	Select to disable the porting processor macro that helps porting code from HCS08 to V1
Enable Porting Support	Select to enable the porting processor macro that helps porting code from HCS08 to V1
None	Select to disable floating point support
Float is IEEE64, double is IEEE64	Select to enable floating point support
No Optimizations	Select to disable optimization level
Easy Debug	Select to enable Level 1 code size optimizations plus register coloring and peephole
Full Optimizations	Select to enable full optimizations

ColdFire Build Options Page for ColdFire V2/3/4

Use this page to enable C++, porting processor macro, hardware startup, and optimization level for ColdFire V2/3/4 derivatives.

NOTE This page will appear only if a ColdFire V2/3/4 derivative or board is selected in the **Devices** page (Figure 2.2).

Figure 2.14 ColdFire Build Options Page — ColdFire V2/3/4


New Bareboard Project
ColdFire Build Options
☐ Enable C++ Support
Hardware Startup
● Full Board Support
○ Minimal Hardware Support
Optimization Level:
● No Optimizations
○ Easy Debug
○ Full Optimizations
< Back
Next >
Finish
Cancel

This project provides full support for the selected board.
The created project provides Standard IO Support through console and terminal window.

No Optimizations

Table 2.12 explains the options available on this page.

Table 2.12 ColdFire 2/3/4 Build Options Page Settings

Option	Description
Enable C++ Support	Check to enable C++ support
Full Board Support	Select to provide full support for the selected board. The created project provides standard input output support through console and terminal window.
Minimal Hardware Support	Select if you do not want to provide board initialization support. The project can be customized or used with the Instruction Set Simulator. The standard input output support is enabled for the Console build target. However, you need to enable UART support for standard input output support through UART, by providing the correct system clock.
None	Select to disable floating point support

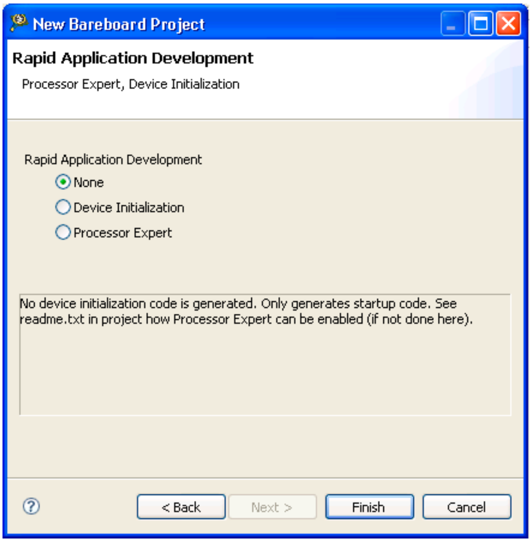
Table 2.12 ColdFire 2/3/4 Build Options Page Settings (*continued*)

Option	Description
Float is IEEE64, double is IEEE64	Select to enable floating point support
No Optimizations	Select to disable optimization level
Easy Debug	Select to enable Level 1 code size optimizations plus register coloring and peephole
Full Optimizations	Select to enable full optimizations

Rapid Application Development Page

Use this page to provide rapid application development (RAD) support when writing your program.

Figure 2.15 Rapid Application Development Page


New Bareboard Project
Rapid Application Development
Processor Expert, Device Initialization
Rapid Application Development
● None
○ Device Initialization
○ Processor Expert
< Back
Next >
Finish
Cancel

No device initialization code is generated. Only generates startup code. See readme.txt in project how Processor Expert can be enabled (if not done here).

Select one of the available RAD options to set up special views in the IDE where you can rapidly configure peripheral devices on the MCU, or pick from a library of field-tested code modules that can implement various device services such as timer interrupts, or a high speed serial interface.

NOTE This page will not appear for the MPC5xx derivatives.

NOTE For more information on how to use the features of the Peripheral Initialization or Processor Expert refer to the *Processor Expert Users Manual* and the *Device Initialization Users Manual*.

Table 2.13 shows the various RAD options available and their purpose.

Table 2.13 RAD Page Settings

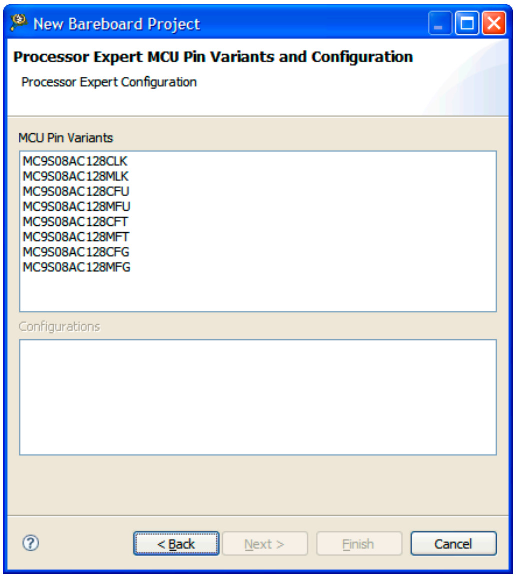
Option	Description
None	No RAD support provided. The wizard's default `startxx.c` file sets up the MCU's stack, its memory management unit (if any) and the C/C++ language's runtime.
Device Initialization	The wizard provides views in the C/C++ Perspective that let you set up the MCU's interrupts and its interrupt vector table. Drivers for the MCU's peripherals are also available.
Processor Expert	The wizard provides views in the C/C++ Perspective that lets you set up the MCU's interrupts, vector table and device initialization. It also provides you with a choice of configurable support modules that implement software services on various MCU peripherals.

NOTE If you select a RAD option other than `None`, the specialized views appear in the C/C++ Perspective after the **New MCU Project** wizard exits.

Processor Expert MCU Pin Variants and Configuration Page

Use this page to provide Processor Expert configuration when creating your project.

Figure 2.16 Processor Expert MCU Pin Variants and Configuration Page



New Bareboard Project
Processor Expert MCU Pin Variants and Configuration
Processor Expert Configuration
MCU Pin Variants

MC9S08AC128CLK
MC9S08AC128MLK
MC9S08AC128CFU
MC9S08AC128MFU
MC9S08AC128CFT
MC9S08AC128MFT
MC9S08AC128CFG
MC9S08AC128MFG

Configurations
< Back
Next >
Finish
Cancel

NOTE The list of variants on this page of the wizard change depending on the selected derivative or board.

NOTE This page will not appear for some ColdFire and all MPC5xx derivatives.

New Linux/uClinux Application Project Wizard

When you start the Microcontrollers **New Linux/uClinux Application Project** wizard, it presents you with a sequence of pages that prompt you for the features and settings to be used when making your program. For example, the devices options lets you select the ColdFire derivative or board you would like to use. Other options let you to specify other settings, such as whether the program executes on an emulator or simulator rather than actual hardware, and the characteristics of the connection that communicates with a hardware target.

This topic describes the various pages that the wizard displays as it assists you in creating a bareboard project. The pages that the wizard presents can differ based upon the option of project type or execution target.

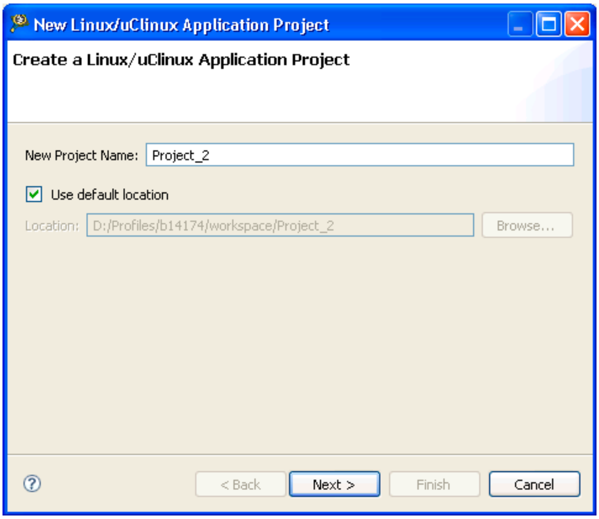
The pages of the **New Linux/uClinux Application Project** wizard are:

- Create a Linux/uClinux Application Project Page
- Device used for Linux Application Debug Page
- Project Language and Output Page
- Connections Page
- Application Debug Options Page

Create a Linux/uClinux Application Project Page

Use this page to specify the project name and the directory where the project files are located.

Figure 2.17 Create a Linux/uClinux Application Project Page



New Linux/uClinux Application Project
Create a Linux/uClinux Application Project

New Project Name: Project_2

☑ Use default location

Location: D:/Profiles/b14174/workspace/Project_2

Browse...
< Back
Next >
Finish
Cancel

Table 2.14 describes the purpose of the various options.

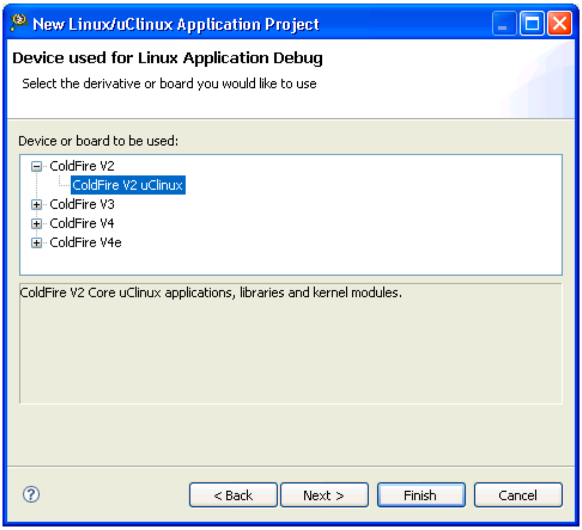
Table 2.14 Create a Linux/uClinux Application Project Page Settings

Option	Description
New Project Name	Enter the name for the new project in this text box.
Use default location	Stores the files required to build the program in the Workbench’s current workspace directory. The project files are located in the directory you specify. Use the **Location** option to select the directory.
Location	Specifies the directory that contains the project files. Click **Browse** to navigate to the desired directory. This option is only available when **Use default location** is clear.

Device used for Linux Application Debug Page

Use this page to select the derivative or board you would like to debug.

Figure 2.18 Device used for Linux Application Debug Page


New Linux/uClinux Application Project
Device used for Linux Application Debug
Select the derivative or board you would like to use
Device or board to be used:
ColdFire V2
ColdFire V2 uClinux
ColdFire V3
ColdFire V4
ColdFire V4e
< Back
Next >
Finish
Cancel

ColdFire V2 Core uClinux applications, libraries and kernel modules.

Table 2.15 describes the purpose of the various options.

Table 2.15 Device used for Linux Application Debug Page

Option	Description
ColdFire V2 > ColdFire V2 uClinux	Select to create ColdFire V2 Core uClinux applications, libraries, and kernel modules.
ColdFire V3 > ColdFire V3 uClinux	Select to create ColdFire V3 Core uClinux applications, libraries, and kernel modules.
ColdFire V4 > ColdFire V4 GNU Linux	Select to create ColdFire V2 Core GNU Linux applications, libraries, and kernel modules.
ColdFire V4e > ColdFire V4e GNU Linux	Select to create ColdFire V4e Core GNU Linux applications, libraries, and kernel modules.

Project Language and Output Page

Use this page to select the programming language that you want to use when writing the program's source code. You can make multiple selections, creating the code in multiple formats.

Figure 2.19 Project Language and Output Page

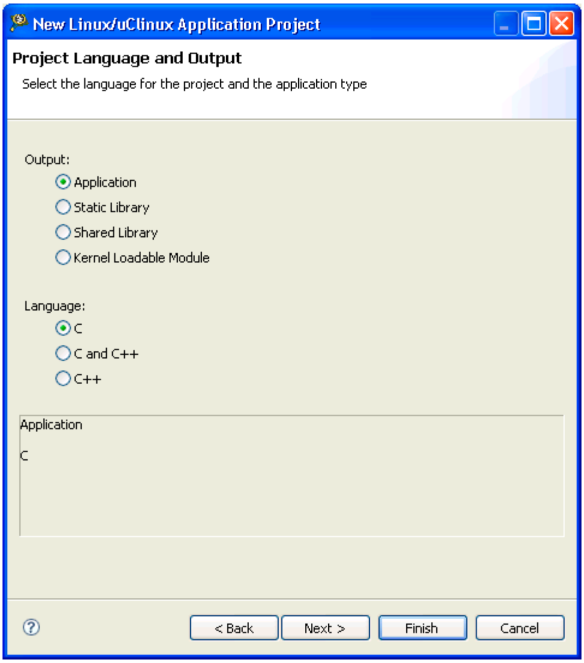

New Linux/uClinux Application Project
Project Language and Output
Select the language for the project and the application type
Output:
● Application
○ Static Library
○ Shared Library
○ Kernel Loadable Module
Language:
● C
○ C and C++
○ C++
Application
C
< Back
Next >
Finish
Cancel

Table 2.16 describes the purpose of the various options.

NOTE Based on your selection, the IDE may show or hide some options.

Table 2.16 Project Language and Output Page Settings

Option	Description
Application	Select if you want the output to be an application. By default, the extension of a loadable module is `.elf`.
Static Library	Select if you want the output to be a static library. By default, the extension of a static library is `.a`.
Shared Library	Select if you want the output to be a shared library. By default, the extension of a shared library is `.so`.

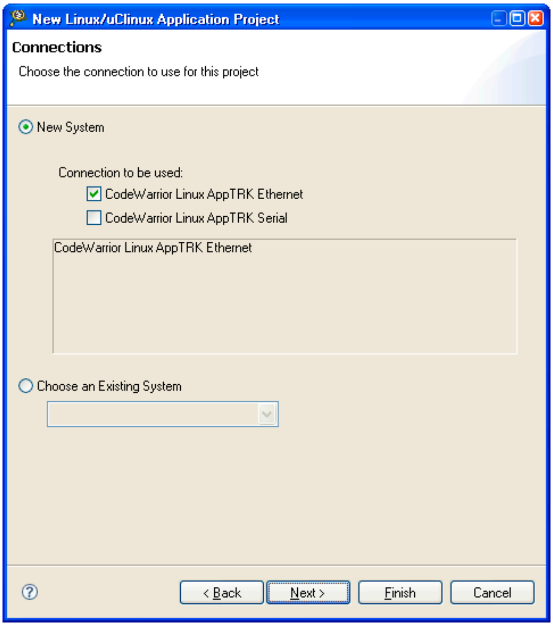
Table 2.16 Project Language and Output Page Settings (*continued*)

Option	Description
Kernel Loadable Module	Select if you want the output to be a kernel loadable module. By default, the extension of a loadable module is `.o`.
C	Select to add C language support.
C and C++	Select to add C and C++ language support. Available for Application and Static options only.
C++	Select to add C++ language support. Available for Application and Static options only.

Connections Page

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

Figure 2.20 Connections Page



Connections

Choose the connection to use for this project

● New System

Connection to be used:

☑ CodeWarrior Linux AppTRK Ethernet

☐ CodeWarrior Linux AppTRK Serial

CodeWarrior Linux AppTRK Ethernet

○ Choose an Existing System

Table 2.17 describes the purpose of the various options.

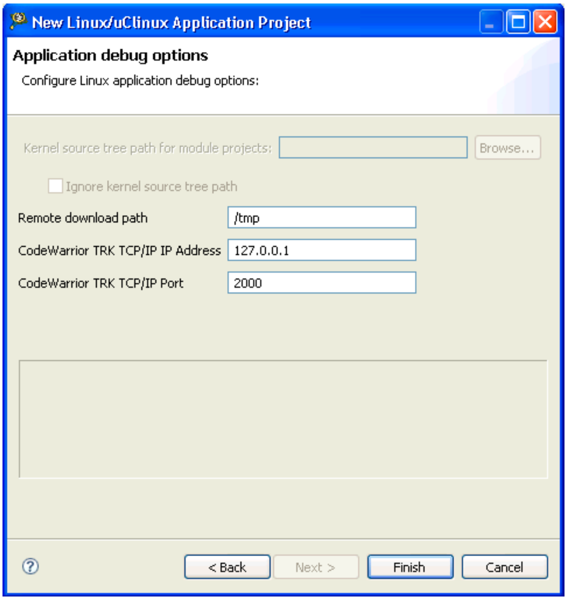
Table 2.17 Connections Page Settings

Option	Description
CodeWarrior Linux App TRK Ethernet	Available only if the Application option is selected on the Project Language and Output page.
CodeWarrior Linux App TRK Serial	Available only if the Application option is selected on the Project Language and Output page.
Choose an Existing System	Chooses an existing remote system configuration. For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

Application Debug Options Page

Use this page to specify the application debug options for a project.

Figure 2.21 Application Debug Options Page



New Linux/uClinux Application Project

Application debug options

Configure Linux application debug options:

Kernel source tree path for module projects:

☐ Ignore kernel source tree path

Remote download path: /tmp

CodeWarrior TRK TCP/IP IP Address: 127.0.0.1

CodeWarrior TRK TCP/IP Port: 2000

Table 2.18 describes the purpose of the various options.

Table 2.18 Application Debug Options Settings

Option	Description
Kernel source tree path for module projects	Click **Browse** to specify or enter the kernel source tree path for module projects.
Ignore kernel source tree path	Select to ignore the kernel source tree path.
Remote download path	Specify the remote download path.
CodeWarrior TRK TCP-IP IP Address	Specify the CodeWarrior TRK TCP/IP Address.
CodeWarrior TRK TCP/IP Port	Specify the CodeWarrior TRK TCP/IP port number.

Creating Projects

The **New Bareboard Project** and **New Linux/uClinux Application Project** wizards help you to quickly create new projects. The wizard generates a project with placeholder files and default settings (build and launch configurations) specific targets. After the project has been created, you can easily change any default setting to suit your needs.

The following topics explain the steps to create Bareboard and Linux/uClinux Application projects for HCS08, RS08, Flexis, ColdFire V1, and ColdFire V2-4e derivatives.

- Creating Bareboard Projects
- Creating Linux/uClinux Application Project

Creating Bareboard Projects

The following topics explain the steps to create bareboard projects for HCS08, RS08, and ColdFire architectures.

- Creating Simulator Projects for HCS08
- Creating Simulator Projects for RS08
- Creating Target Board Project for ColdFire V1
- Creating Target Board Project for ColdFire V2-4e
- Creating Target Board Project for MPC56xx
- Creating Target Board Project for Kinetis

NOTE The ColdFire V1, ColdFire V2, ColdFire V3 ColdFire V4, ColdFire V4e, and ColdFire Evaluation Boards architectures do not support Full Chip Simulation.

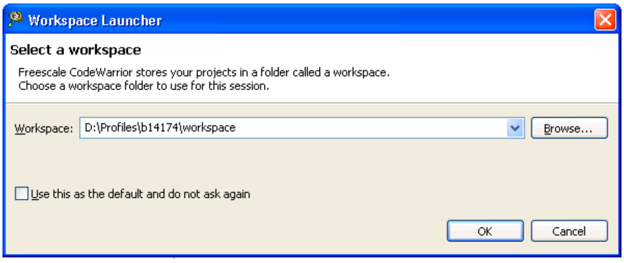
Creating Simulator Projects for HCS08

To create a simulator project for HCS08 using the **New Bareboard Project** wizard**,** perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.22 WorkSpace Launcher Dialog Box



Workspace Launcher

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

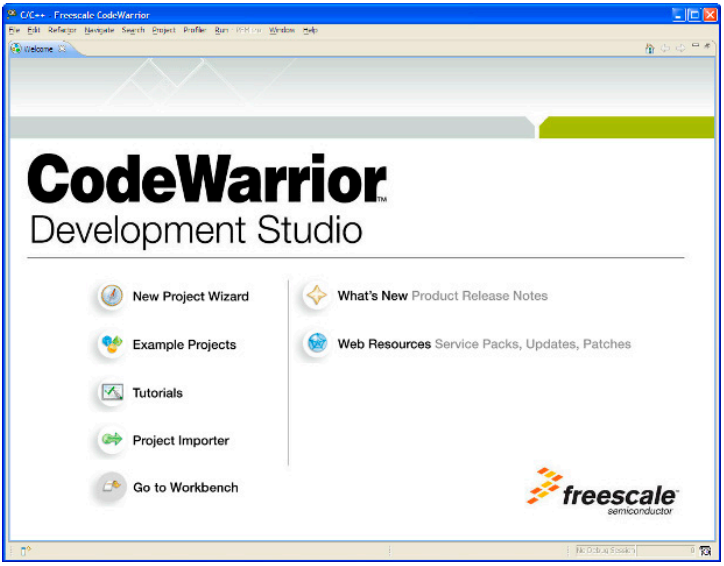
☐ Use this as the default and do not ask again

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.23 Welcome Page


C/C++ - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run Window Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale semiconductor

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

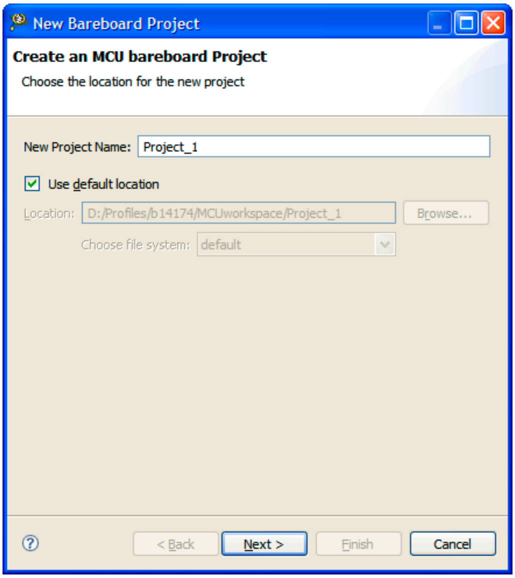
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears (Figure 2.24).

5. Specify a name for the new project. For example, enter the project name as `Project_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.24 New Bareboard Project Wizard — Create an MCU Bareboard Project Page



New Bareboard Project
Create an MCU bareboard Project
Choose the location for the new project

New Project Name: Project_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/Project_1

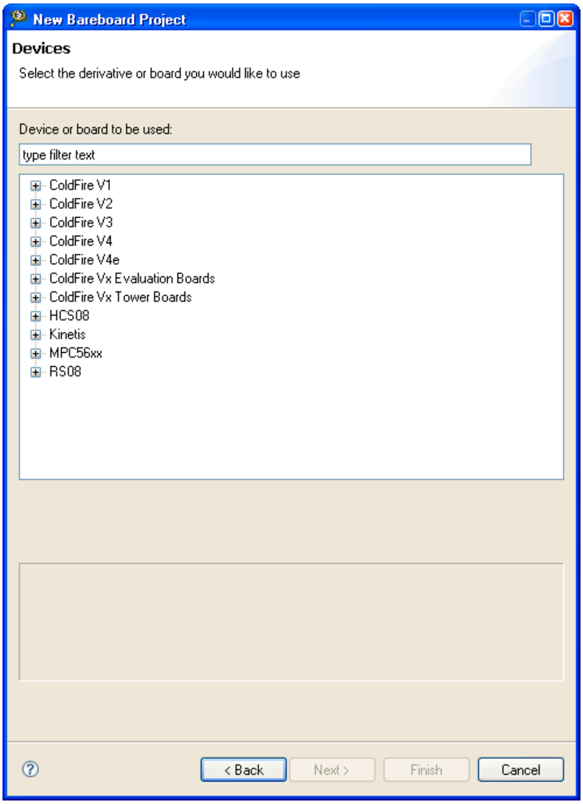
Choose file system: default

Browse...
< Back
Next >
Finish
Cancel

6. Click **Next**.

   The **Devices** page appears (Figure 2.25).

Figure 2.25 New Bareboard Project Wizard — Devices Page



New Bareboard Project

Devices

Select the derivative or board you would like to use

Device or board to be used:

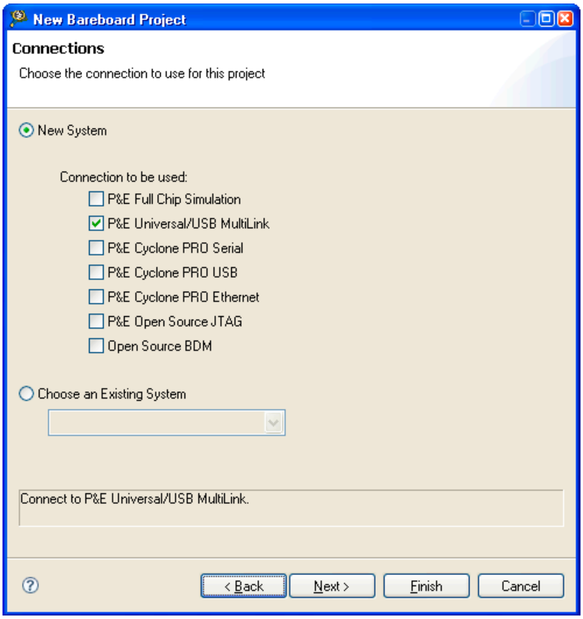
type filter text

- ColdFire V1
- ColdFire V2
- ColdFire V3
- ColdFire V4
- ColdFire V4e
- ColdFire Vx Evaluation Boards
- ColdFire Vx Tower Boards
- HCS08
- Kinetis
- MPC56xx
- RS08

7. Expand the tree control and select the derivative or board you would like to use. For example, select `HCS08 > HCS08A Family > MC9S08AC128`.
8. Click **Next**.

   The **Connections** page appears.

Figure 2.26 New Bareboard Project Wizard — Connections Page



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Full Chip Simulation
☑ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ P&E Open Source JTAG
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

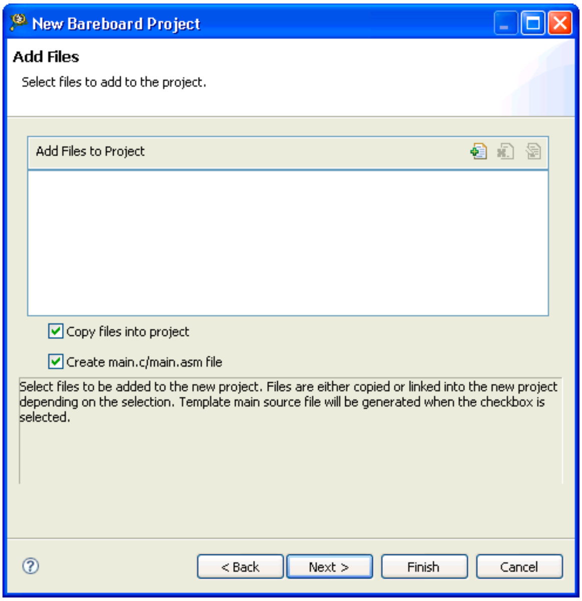
9. Check the **P&E Universal?USB MultiLink** checkbox.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

10. Click **Next**.

    The **Add Files** page appears.

Figure 2.27 New Bareboard Project Wizard — Add Files Page



New Bareboard Project
Add Files
Select files to add to the project.
Add Files to Project
< Back
Next >
Finish
Cancel

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

11. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.

    b. Click **OK** to close the **Add File Path** dialog box.

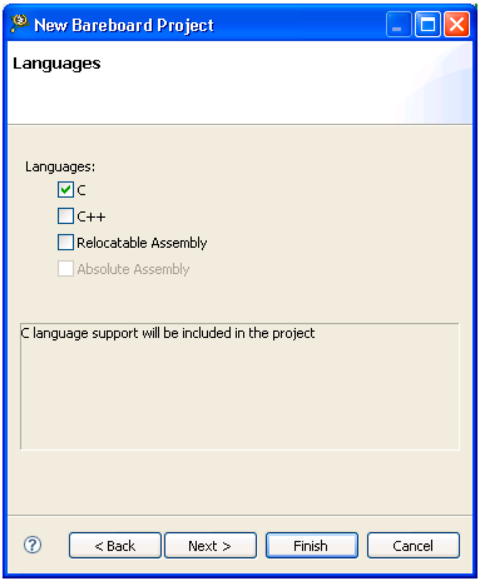
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **Languages** page appears.

Figure 2.28 New Bareboard Project Wizard — Languages Page



Languages:
☑ C
☐ C++
☐ Relocatable Assembly
☐ Absolute Assembly

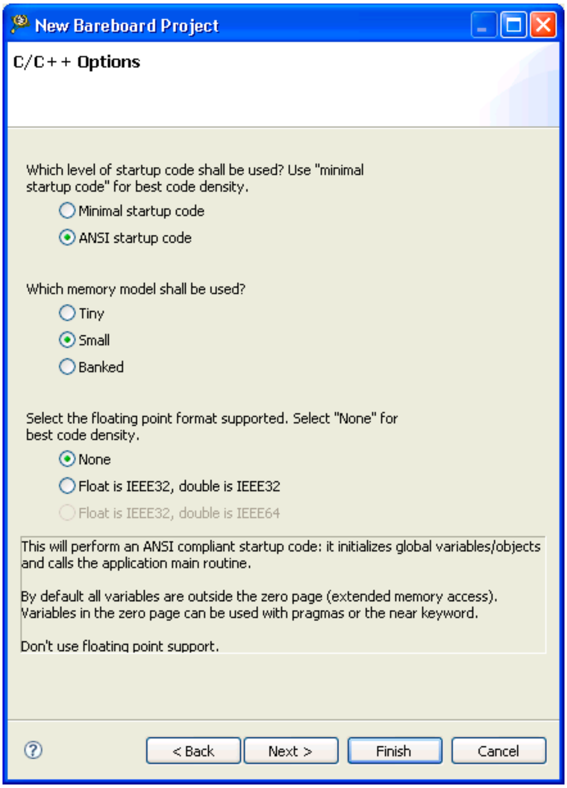
C language support will be included in the project

13. Select the programming language you want to use. For example, check the **C** checkbox.
14. Click **Next**.

    The **C/C++ Options** page appears.

NOTE If you check only the **Relocatable Assembly** or **Absolute Assembly** checkbox, clicking **Next** will display the **Connections** page instead (Figure 2.29).

Figure 2.29 New Bareboard Project Wizard — C/C++ Options Page



Which level of startup code shall be used? Use "minimal startup code" for best code density.

○ Minimal startup code
● ANSI startup code

Which memory model shall be used?

○ Tiny
● Small
○ Banked

Select the floating point format supported. Select "None" for best code density.

● None
○ Float is IEEE32, double is IEEE32
○ Float is IEEE32, double is IEEE64

This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.

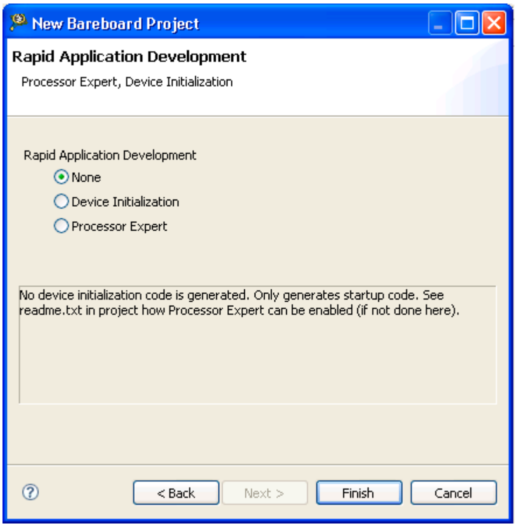
By default all variables are outside the zero page (extended memory access). Variables in the zero page can be used with pragmas or the near keyword.

Don't use floating point support.

15. Select the appropriate level of startup code, memory model, and floating point format.
16. Click **Next**.

    The **Rapid Application Development** page appears.

Figure 2.30 New Bareboard Project Wizard — Rapid Application Development Page



New Bareboard Project
Rapid Application Development
Processor Expert, Device Initialization
Rapid Application Development
● None
○ Device Initialization
○ Processor Expert
< Back
Next >
Finish
Cancel

No device initialization code is generated. Only generates startup code. See readme.txt in project how Processor Expert can be enabled (if not done here).

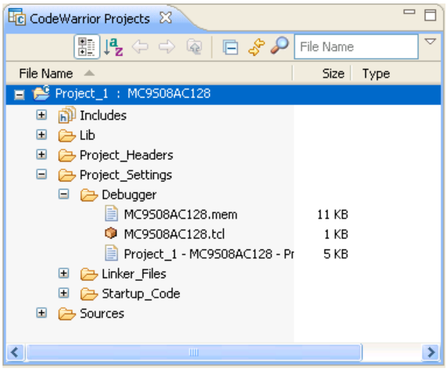
17. Select the appropriate option to support rapid application development.
    - **None** — Select to generate only startup code.
    - **Device Initialization** — Select to generate the initialization code for on-chip peripherals, interrupt vector table, and template for interrupt vector service routines.
    - **Processor Expert** — Select to generate the device initialization code, including low-level drivers.

NOTE If you select the **Processor Expert** option, clicking the Next button will display the **Processor Expert MCU Pin Variants and Configuration** page. Here you can select the required microcontroller variant and its configuration.

18. Click **Finish**.

    The wizard creates a simulator project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.31 CodeWarrior Projects View



File Name	Size	Type
Project_1 : MC9S08AC128		
Includes		
Lib		
Project_Headers		
Project_Settings		
Debugger		
MC9S08AC128.mem	11 KB	
MC9S08AC128.tcl	1 KB	
Project_1 - MC9S08AC128 - Pr	5 KB	
Linker_Files		
Startup_Code		
Sources		

The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

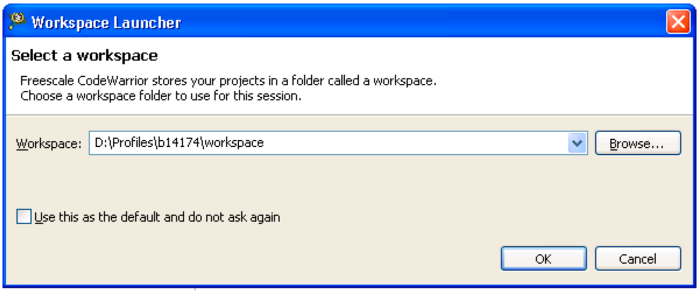
Creating Simulator Projects for RS08

To create a simulator project for RS08 using the **New Bareboard Project** wizard**,** perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.32 WorkSpace Launcher Dialog Box



Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

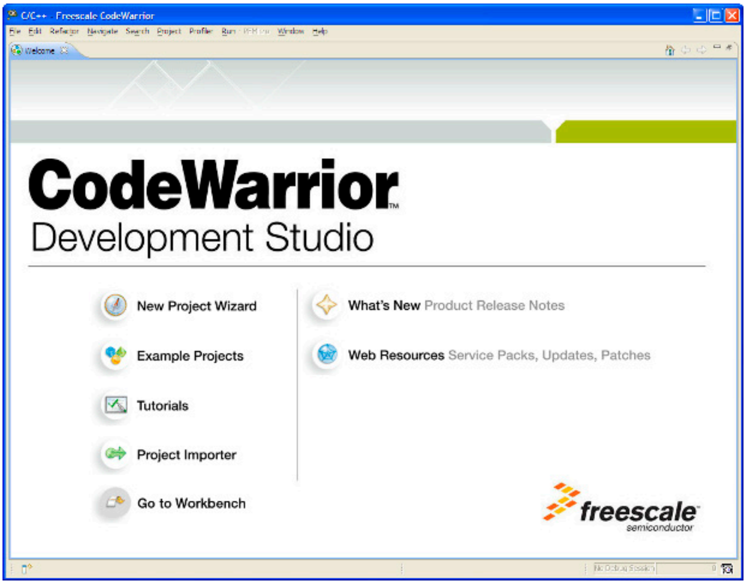
☐ Use this as the default and do not ask again

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.33 Welcome Page


C/C++ - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run Window Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale semiconductor

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

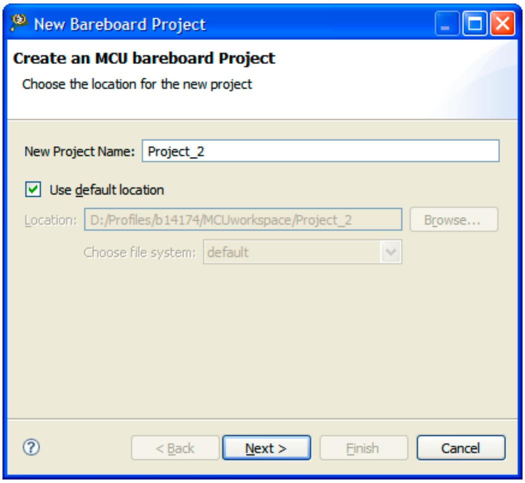
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project page** of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `Project_2`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.34 New Bareboard Project Wizard — Create an MCU Bareboard Project Page


New Bareboard Project
Create an MCU bareboard Project
Choose the location for the new project

New Project Name: Project_2

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/Project_2

Choose file system: default

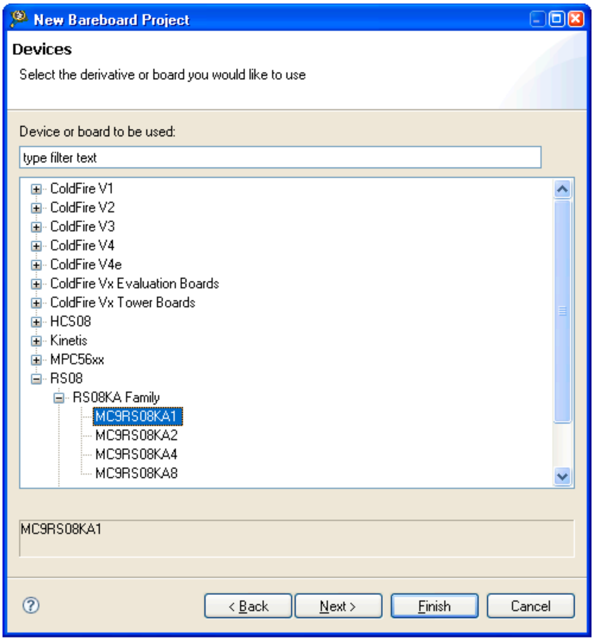
Browse...
< Back
Next >
Finish
Cancel

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select `RS08 > RS08KA Family > MC9RS08KA1`.

Figure 2.35 New Bareboard Project Wizard — Devices Page



New Bareboard Project
Devices
Select the derivative or board you would like to use
Device or board to be used:
type filter text
ColdFire V1
ColdFire V2
ColdFire V3
ColdFire V4
ColdFire V4e
ColdFire Vx Evaluation Boards
ColdFire Vx Tower Boards
HCS08
Kinetis
MPC56xx
RS08
RS08KA Family
MC9RS08KA1
MC9RS08KA2
MC9RS08KA4
MC9RS08KA8
MC9RS08KA1
< Back
Next >
Finish
Cancel

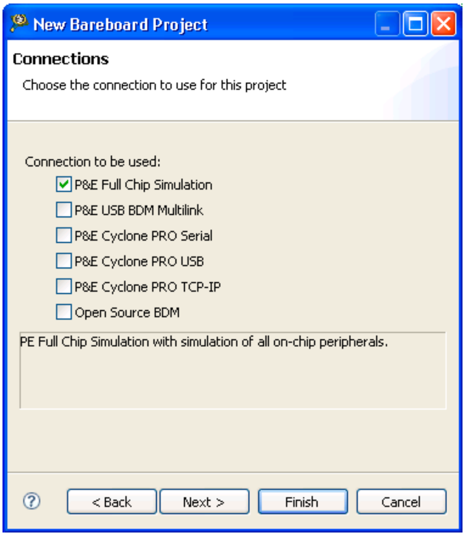
8. Click **Next**.

   The **Connections** page appears.

9. Check the **P&E Full Chip Simulation** checkbox.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

Figure 2.36 New Bareboard Project Wizard — Connections Page


New Bareboard Project
Connections
Choose the connection to use for this project

Connection to be used:

- ☑ P&E Full Chip Simulation
- ☐ P&E USB BDM Multilink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO TCP-IP
- ☐ Open Source BDM

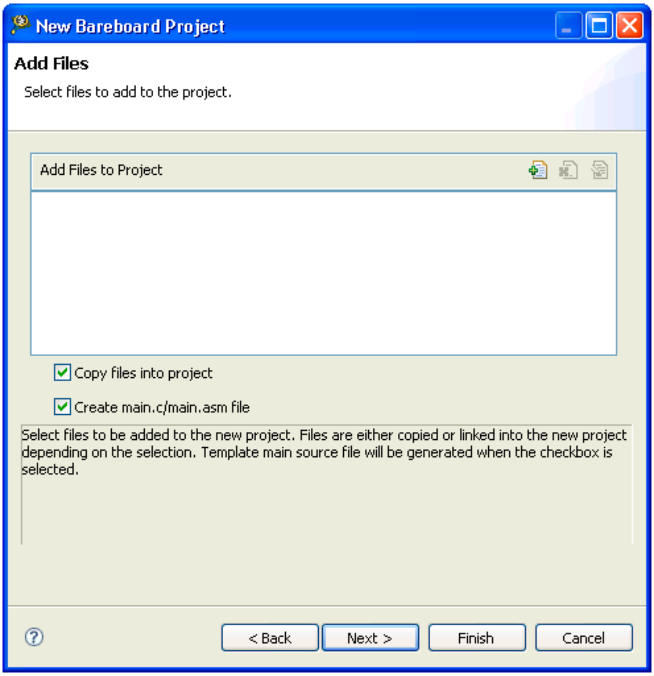
PE Full Chip Simulation with simulation of all on-chip peripherals.

< Back
Next >
Finish
Cancel

10. Click **Next**.

    The **Add Files** page appears.

Figure 2.37 New Bareboard Project Wizard — Add Files Page



Add Files

Select files to add to the project.

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

11. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.

    b. Click **OK** to close the **Add File Path** dialog box.

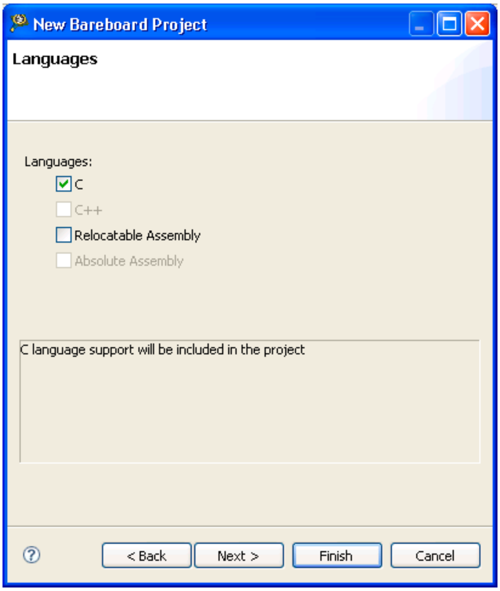
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **Languages** page appears.

Figure 2.38 New Bareboard Project Wizard — Languages Page



Languages:
☑ C
☐ C++
☐ Relocatable Assembly
☐ Absolute Assembly

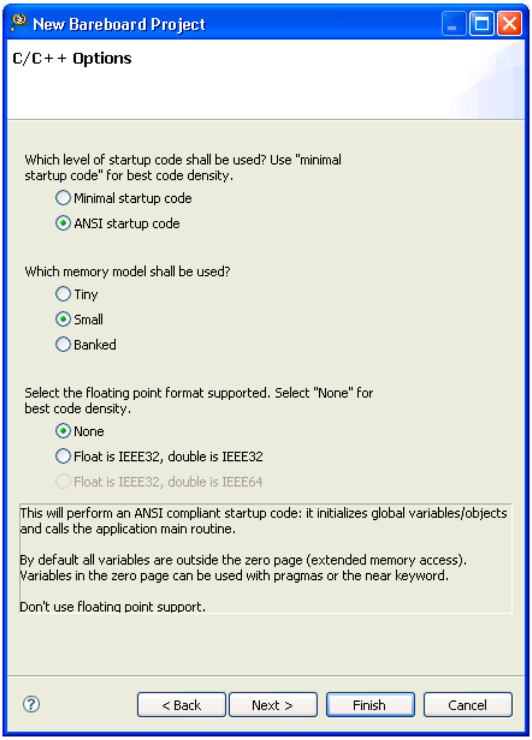
C language support will be included in the project

13. Select the programming language you want to use. For example, check the **C** checkbox.
14. Click **Next**.

    The **C/C++ Options** page appears.

NOTE If you check only the **Relocatable Assembly** or **Absolute Assembly** checkbox, clicking **Next** will display the **Connections** page instead (Figure 2.36).

Figure 2.39 New Bareboard Project Wizard — C/C++ Options Page



Which level of startup code shall be used? Use "minimal startup code" for best code density.

○ Minimal startup code
● ANSI startup code

Which memory model shall be used?

○ Tiny
● Small
○ Banked

Select the floating point format supported. Select "None" for best code density.

● None
○ Float is IEEE32, double is IEEE32
○ Float is IEEE32, double is IEEE64

This will perform an ANSI compliant startup code: it initializes global variables/objects and calls the application main routine.

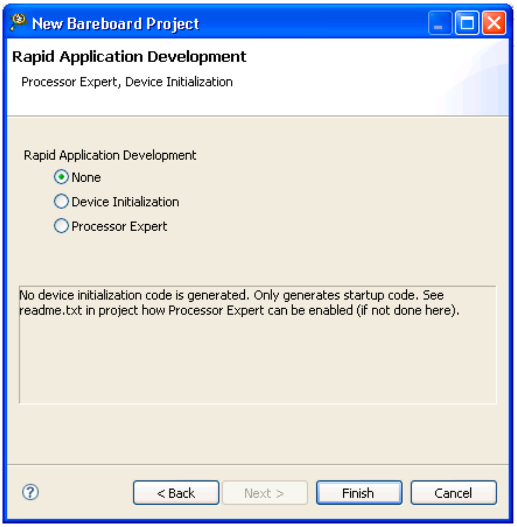
By default all variables are outside the zero page (extended memory access). Variables in the zero page can be used with pragmas or the near keyword.

Don't use floating point support.

15. Select the appropriate level of startup code, memory model, and floating point format.
16. Click **Next**.

    The **Rapid Application Development** page appears.

Figure 2.40 New Bareboard Project Wizard — Rapid Application Development Page


New Bareboard Project
Rapid Application Development
Processor Expert, Device Initialization
Rapid Application Development
● None
○ Device Initialization
○ Processor Expert
< Back
Next >
Finish
Cancel

No device initialization code is generated. Only generates startup code. See readme.txt in project how Processor Expert can be enabled (if not done here).

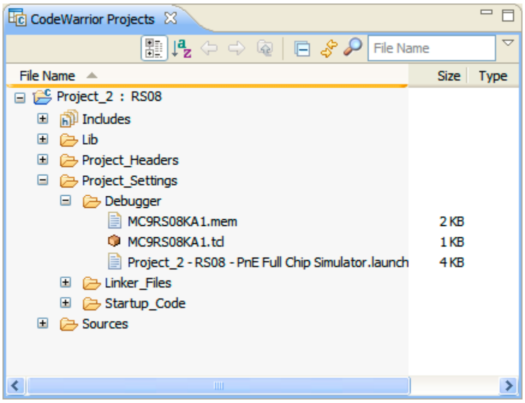
17. Select the appropriate option to support rapid application development.
    - **None** — Select to generate only startup code.
    - **Device Initialization** — Select to generate the initialization code for on-chip peripherals, interrupt vector table, and template for interrupt vector service routines.
    - **Processor Expert** — Select to generate the device initialization code, including low-level drivers.

NOTE If you select the **Processor Expert** option, clicking the Next button will display the **Processor Expert MCU Pin Variants and Configuration** page. Here you can select the required microcontroller variant and its configuration.

18. Click **Finish**.

    The wizard creates a simulator project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.41 CodeWarrior Projects View


CodeWarrior Projects
File Name
File Name
Size
Type
Project_2 : RS08
Includes
Lib
Project_Headers
Project_Settings
Debugger
MC9RS08KA1.mem
2 KB
MC9RS08KA1.tcl
1 KB
Project_2 - RS08 - PnE Full Chip Simulator.launch
4 KB
Linker_Files
Startup_Code
Sources

The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

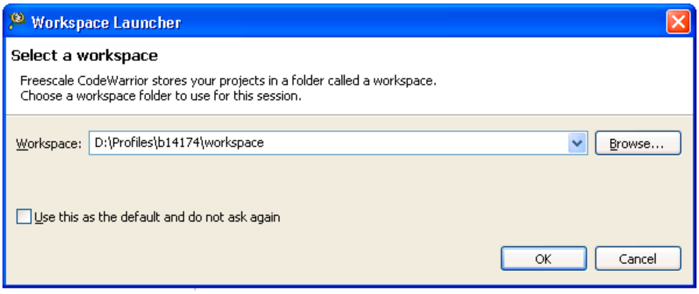
Creating Target Board Project for ColdFire V1

To create a target board project for ColdFire V1 using the **New Bareboard Project** wizard**,** perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.42 WorkSpace Launcher Dialog Box


Workspace Launcher
Browse...
OK
Cancel

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

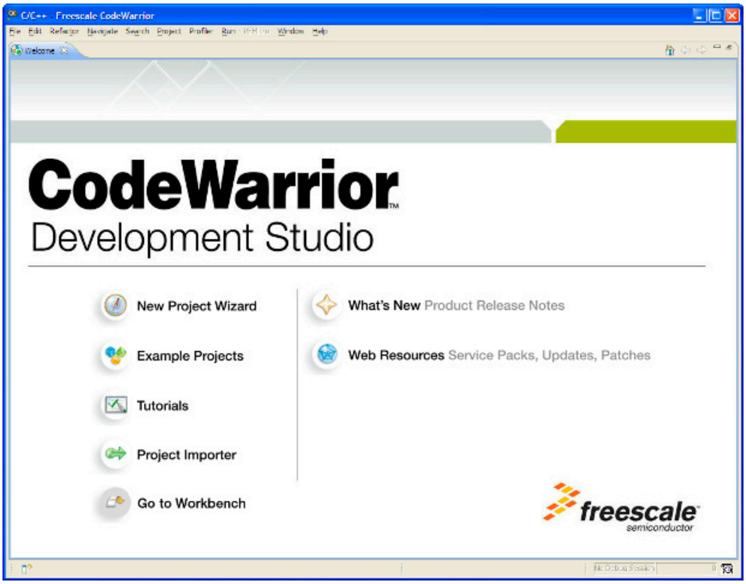
☐ Use this as the default and do not ask again

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.43 Welcome Page


C/C++ - Freescale CodeWarrior
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale
semiconductor

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

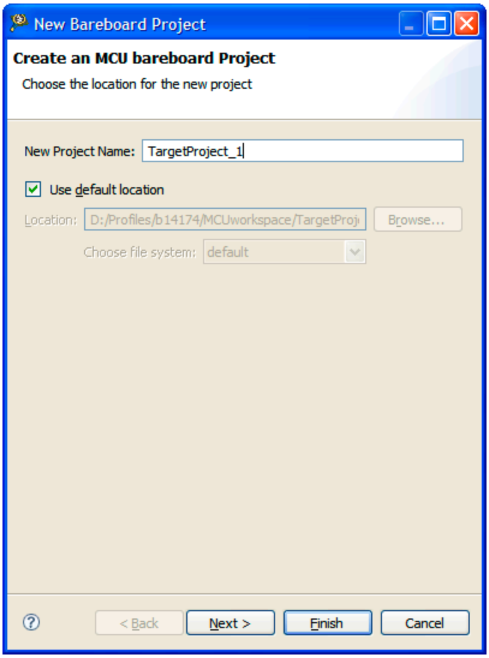
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `TargetProject_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.44 New Bareboard Project Wizard — Create an MCU Bareboard Project Page


New Bareboard Project
Create an MCU bareboard Project
Choose the location for the new project

New Project Name: TargetProject_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/TargetProj

Choose file system: default

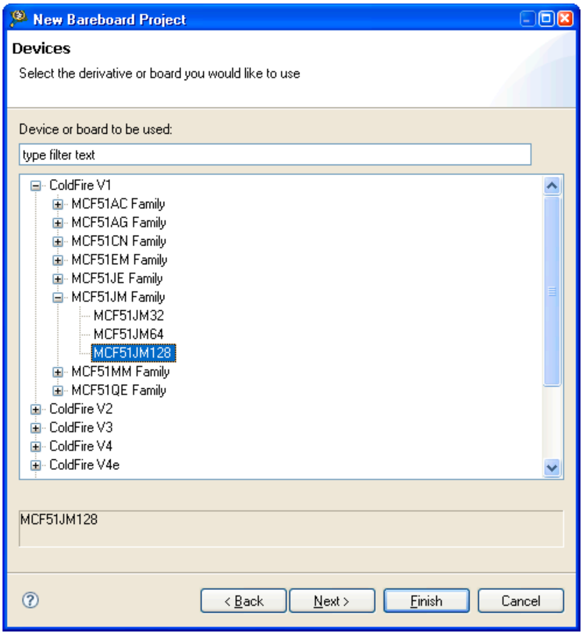
Browse...
< Back
Next >
Finish
Cancel

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select `ColdFire V1 > MCF51JM Family > MCF51JM128`.

Figure 2.45 New Bareboard Project Wizard — Devices Page



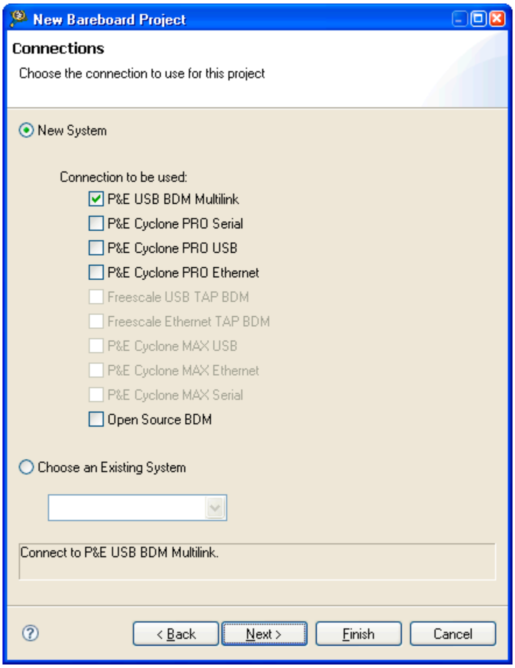
New Bareboard Project
Devices
Select the derivative or board you would like to use
Device or board to be used:
type filter text
ColdFire V1
MCF51AC Family
MCF51AG Family
MCF51CN Family
MCF51EM Family
MCF51JE Family
MCF51JM Family
MCF51JM32
MCF51JM64
MCF51JM128
MCF51MM Family
MCF51QE Family
ColdFire V2
ColdFire V3
ColdFire V4
ColdFire V4e
MCF51JM128
< Back
Next >
Finish
Cancel

8. Click **Next**.

   The **Connections** page appears.

9. Check the appropriate connection. For example, check the **P&E USB BDM MultiLink** checkbox.

Figure 2.46 New Bareboard Project Wizard — Connections Page



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☑ P&E USB BDM Multilink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☐ Open Source BDM

○ Choose an Existing System

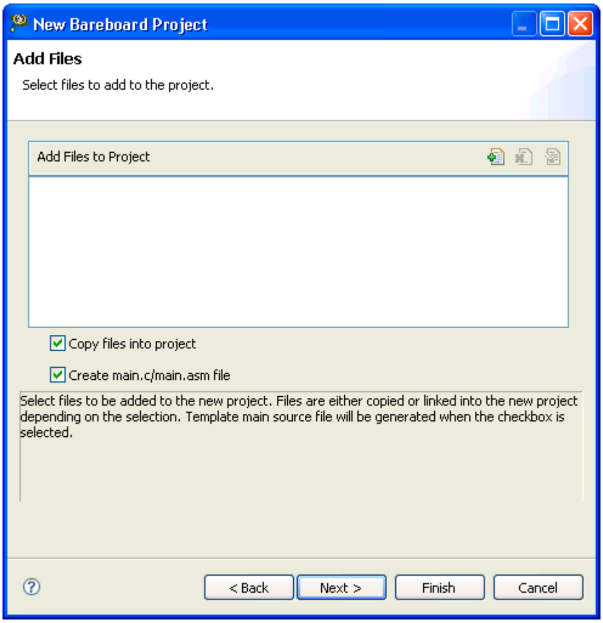
Connect to P&E USB BDM Multilink.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

10. Click **Next**.

    The **Add Files** page appears.

Figure 2.47 New Bareboard Project Wizard — Add Files Page



Select files to add to the project.

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

11. If you want to add a file to the project, click .

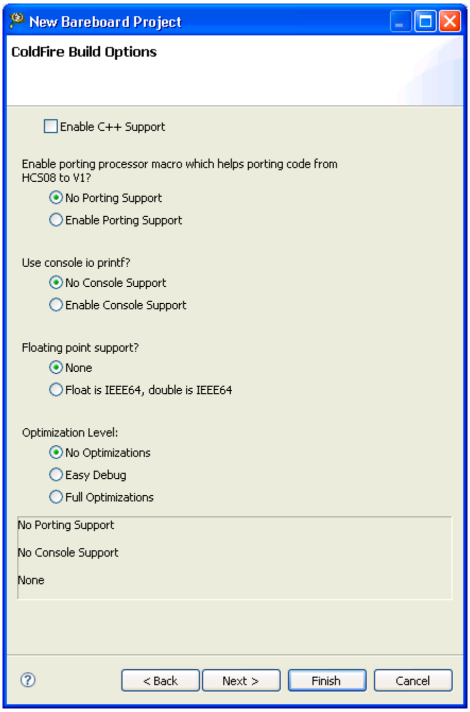
    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.
    b. Click **OK** to close the **Add File Path** dialog box.
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.
    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **ColdFire Build Options** page appears.

Figure 2.48 New Bareboard Project Wizard — ColdFire Build Options Page



ColdFire Build Options

☐ Enable C++ Support

Enable porting processor macro which helps porting code from HCS08 to V1?
- ● No Porting Support
- ○ Enable Porting Support

Use console io printf?
- ● No Console Support
- ○ Enable Console Support

Floating point support?
- ● None
- ○ Float is IEEE64, double is IEEE64

Optimization Level:
- ● No Optimizations
- ○ Easy Debug
- ○ Full Optimizations

No Porting Support

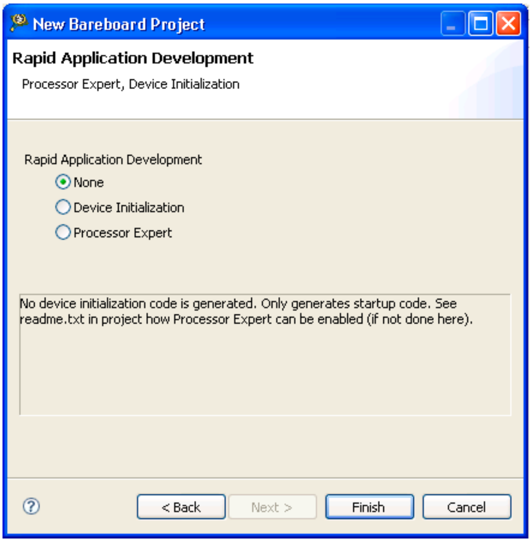
No Console Support

None

13. Select the appropriate options to enable C++, porting processor macro, console, and floating point supports.
14. Click **Next**.

    The **Rapid Application Development** page appears.

Figure 2.49 New Bareboard Project Wizard — Rapid Application Development Page



Rapid Application Development

Processor Expert, Device Initialization

Rapid Application Development

● None

○ Device Initialization

○ Processor Expert

No device initialization code is generated. Only generates startup code. See readme.txt in project how Processor Expert can be enabled (if not done here).

15. Select the appropriate option to support rapid application development.
    - **None** — Select to generate only startup code.
    - **Device Initialization** — Select to generate the initialization code for on-chip peripherals, interrupt vector table, and template for interrupt vector service routines.
    - **Processor Expert** — Select to generate the device initialization code, including low-level drivers.

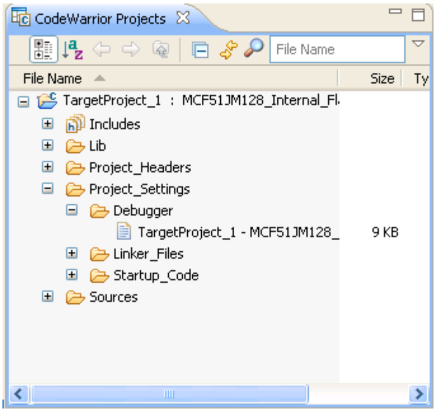
NOTE If you select the **Processor Expert** option, clicking the Next button will display the **Processor Expert MCU Pin Variants and Configuration** page. Here you can select the required microcontroller variant and its configuration.

---

16. Click **Finish**.

    The wizard creates a project for the ColdFire V1 architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.50 CodeWarrior Projects View


CodeWarrior Projects
File Name
File Name
Size
Ty
TargetProject_1 : MCF51JM128_Internal_Fl
Includes
Lib
Project_Headers
Project_Settings
Debugger
TargetProject_1 - MCF51JM128_
9 KB
Linker_Files
Startup_Code
Sources

The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

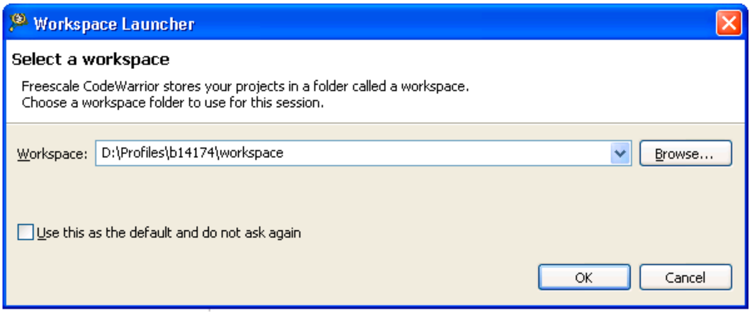
Creating Target Board Project for ColdFire V2-4e

To create a project for ColdFire V2-4e using the **New Bareboard Project** wizard**,** perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.51 WorkSpace Launcher Dialog Box


Workspace Launcher

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

☐ Use this as the default and do not ask again

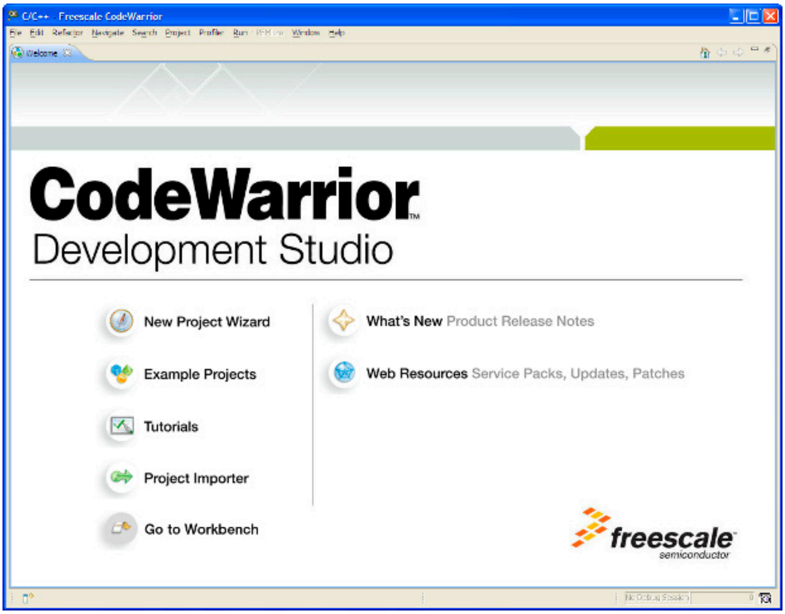
Browse...
OK
Cancel

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.52 Welcome Page


C/C++ - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run Window Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale semiconductor

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

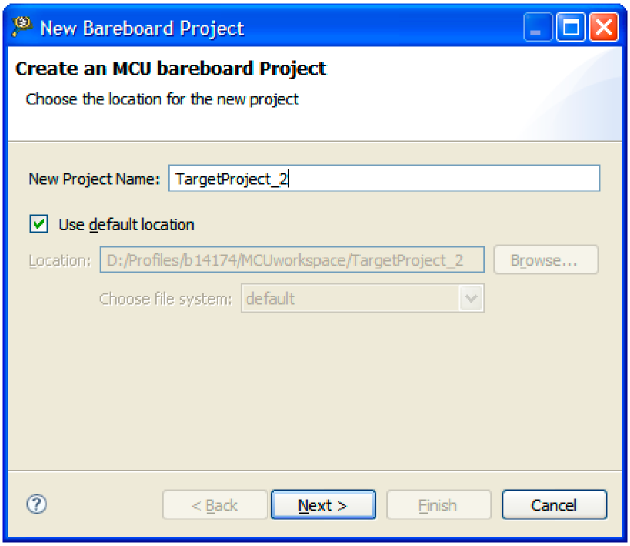
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `TargetProject_2`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.53 New Bareboard Project Wizard — Create an MCU Bareboard Project Page



New Bareboard Project
Create an MCU bareboard Project
Choose the location for the new project
Browse...
< Back
Next >
Finish
Cancel

New Project Name: TargetProject_2

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/TargetProject_2

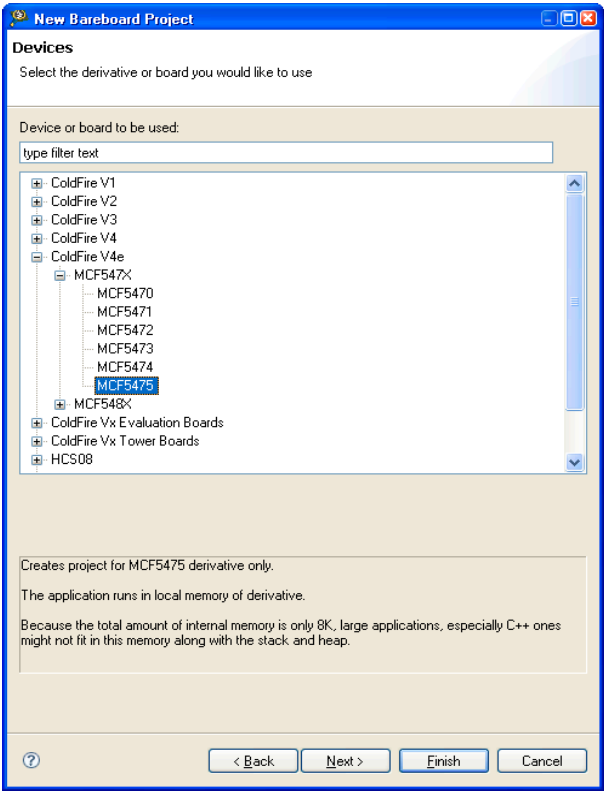
Choose file system: default

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select `ColdFire V4e > MCF547X > MCF5475`.

Figure 2.54 New Bareboard Project Wizard — Devices



New Bareboard Project
Devices
Select the derivative or board you would like to use
Device or board to be used:
type filter text
ColdFire V1
ColdFire V2
ColdFire V3
ColdFire V4
ColdFire V4e
MCF547X
MCF5470
MCF5471
MCF5472
MCF5473
MCF5474
MCF5475
MCF548X
ColdFire Vx Evaluation Boards
ColdFire Vx Tower Boards
HCS08
< Back
Next >
Finish
Cancel

Creates project for MCF5475 derivative only.

The application runs in local memory of derivative.

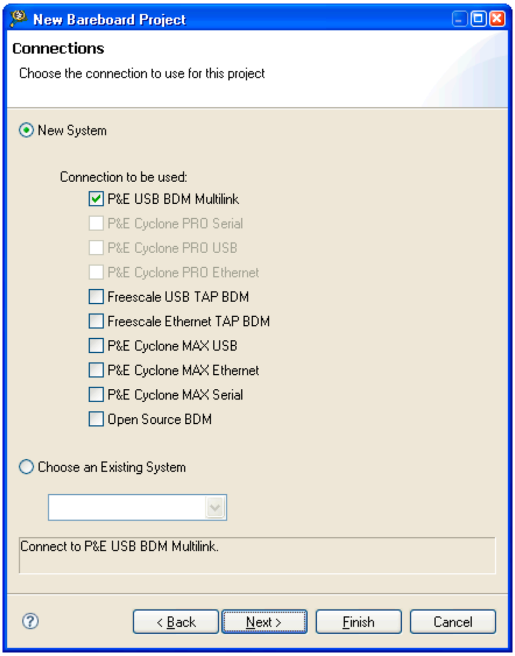
Because the total amount of internal memory is only 8K, large applications, especially C++ ones might not fit in this memory along with the stack and heap.

8. Click **Next**.

   The **Connections** page appears.

9. Check the appropriate connection. For example, check the **P&E USB BDM MultiLink** checkbox.

Figure 2.55 New Bareboard Project Wizard — Connections Page



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☑ P&E USB BDM Multilink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☐ Open Source BDM

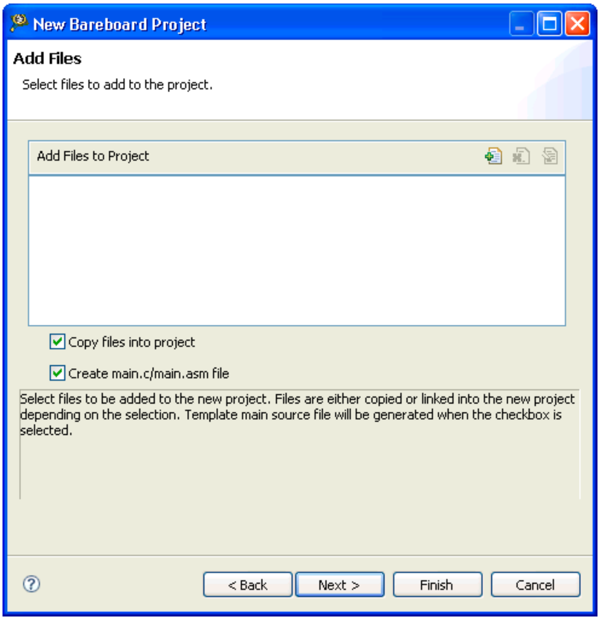
○ Choose an Existing System

Connect to P&E USB BDM Multilink.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

The **Add Files** page appears.

Figure 2.56 New Bareboard Project Wizard — Add Files Page



Select files to add to the project.

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

10. If you want to add a file to the project, click .

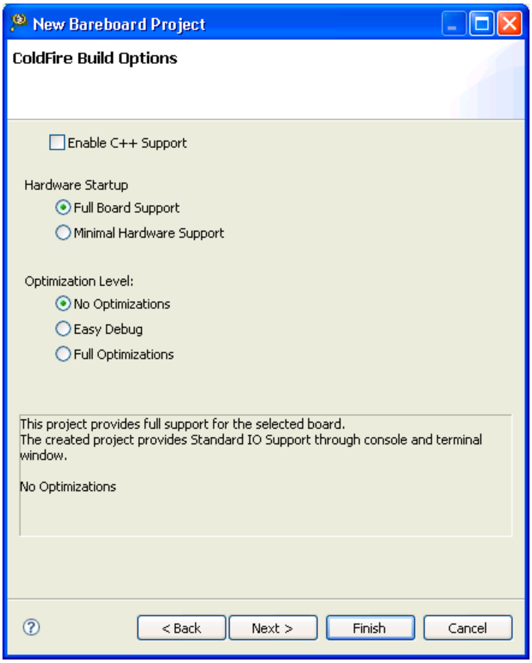
    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.
    b. Click **OK** to close the **Add File Path** dialog box.
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.
    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

11. Click **Next**.

    The **ColdFire Build Options** page appears.

Figure 2.57 New Bareboard Project Wizard — ColdFire Build Options Page



☐ Enable C++ Support

Hardware Startup
● Full Board Support
○ Minimal Hardware Support

Optimization Level:
● No Optimizations
○ Easy Debug
○ Full Optimizations

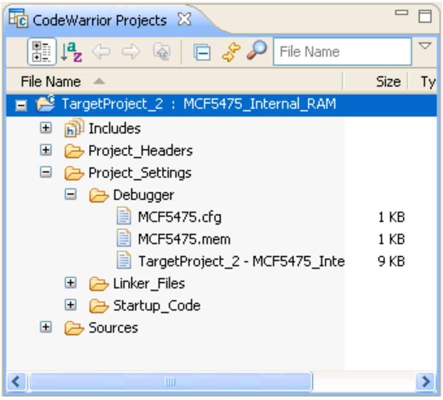
This project provides full support for the selected board.
The created project provides Standard IO Support through console and terminal window.

No Optimizations

12. Select the appropriate options to enable C++, porting processor macro, console, and floating point supports.
13. Click **Finish**.

    The wizard creates a project for the ColdFire V4e architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.58 CodeWarrior Projects View



- TargetProject_2 : MCF5475_Internal_RAM
  - Includes
  - Project_Headers
  - Project_Settings
    - Debugger
      - MCF5475.cfg 1 KB
      - MCF5475.mem 1 KB
      - TargetProject_2 - MCF5475_Inte 9 KB
    - Linker_Files
    - Startup_Code
  - Sources

The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

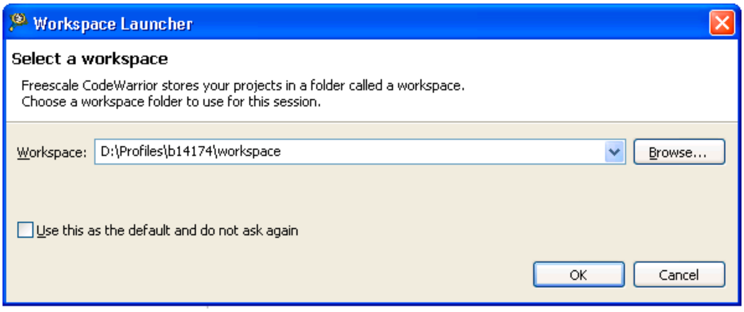
Creating Target Board Project for MPC56xx

To create a project for MPC56xx using the **New Bareboard Project** wizard, perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.59 WorkSpace Launcher Dialog Box



Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

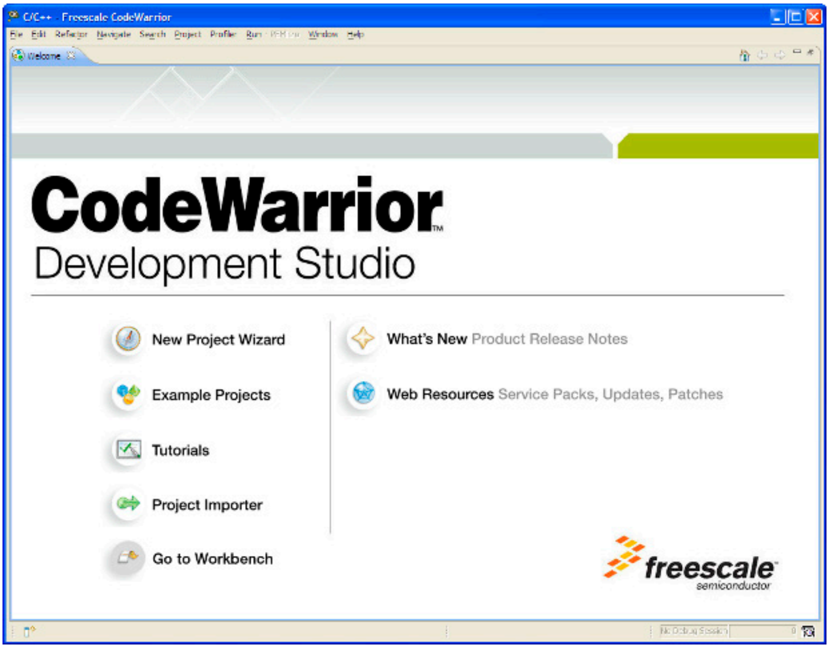
☐ Use this as the default and do not ask again

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.60 Welcome Page


C/C++ - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run Window Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale semiconductor
No Debug Session

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

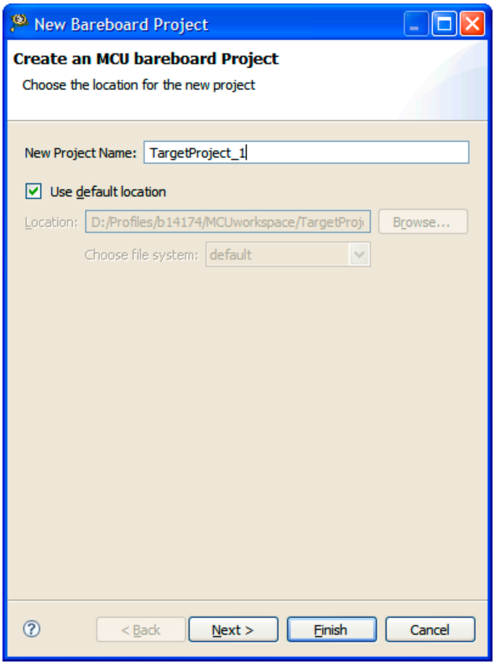
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `TargetProject_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.61 New Bareboard Project Wizard — Create an MCU Bareboard Project Page



New Bareboard Project
Create an MCU bareboard Project
Choose the location for the new project

New Project Name: TargetProject_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/TargetProj

Choose file system: default

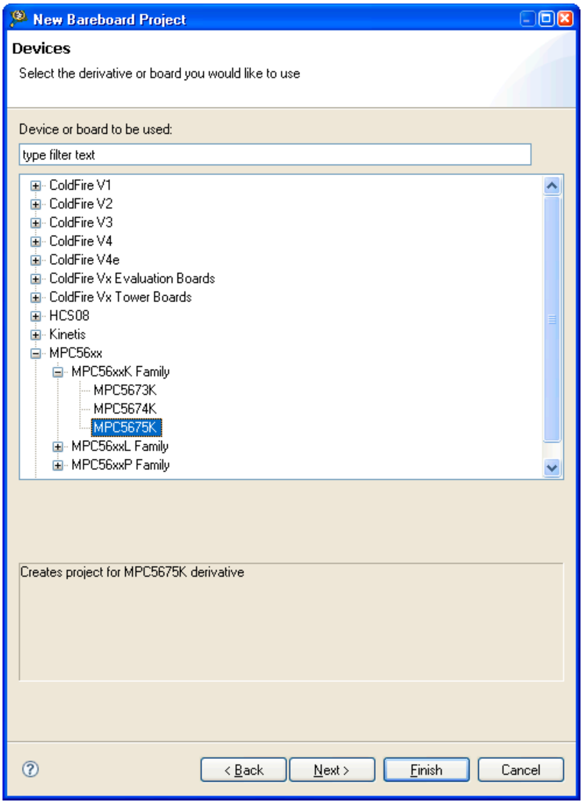
Browse...
< Back
Next >
Finish
Cancel

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxK Family > MPC5675K**.

Figure 2.62 New Bareboard Project Wizard — Devices


New Bareboard Project
Devices
Select the derivative or board you would like to use
Device or board to be used:
type filter text
ColdFire V1
ColdFire V2
ColdFire V3
ColdFire V4
ColdFire V4e
ColdFire Vx Evaluation Boards
ColdFire Vx Tower Boards
HCS08
Kinetis
MPC56xx
MPC56xxK Family
MPC5673K
MPC5674K
MPC5675K
MPC56xxL Family
MPC56xxP Family
< Back
Next >
Finish
Cancel

Creates project for MPC5675K derivative

NOTE The MPC56xxL (Leopard) derivative currently supported in CodeWarrior MPC55xx/MPC56xx v2.x is MPC5643L.

NOTE The MPC56xxP (Pictus) derivatives supported in CodeWarrior MPC55xx/MPC56xx v2.x are: MPC5601P *M07N, MPC5602P *M07N, MPC5603P *M07N, MPC5604P *M07N, MPC5601P *M26V, MPC5602P *M26V, MPC5603P *M26V, and MPC5604P *M26V. However, not all derivatives are supported in this build.

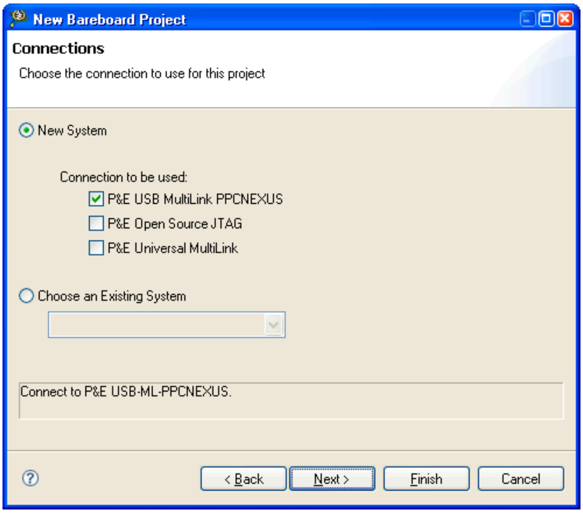
NOTE The MPC567xK (Komodo) derivatives supported in CW MPC55xx/MPC56xx v2.x are: MPC5671K, MPC5672K, MPC5673K, MPC5674K, and MPC5675K. However, not all derivatives are supported in this build.

8. Click **Next**.

   The **Connections** page appears.

9. Check the appropriate connection. For example, check the **P&E USB MultiLink PPCNEXUS** checkbox.

Figure 2.63 New Bareboard Project Wizard — Connections Page



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☑ P&E USB MultiLink PPCNEXUS
☐ P&E Open Source JTAG
☐ P&E Universal MultiLink

○ Choose an Existing System

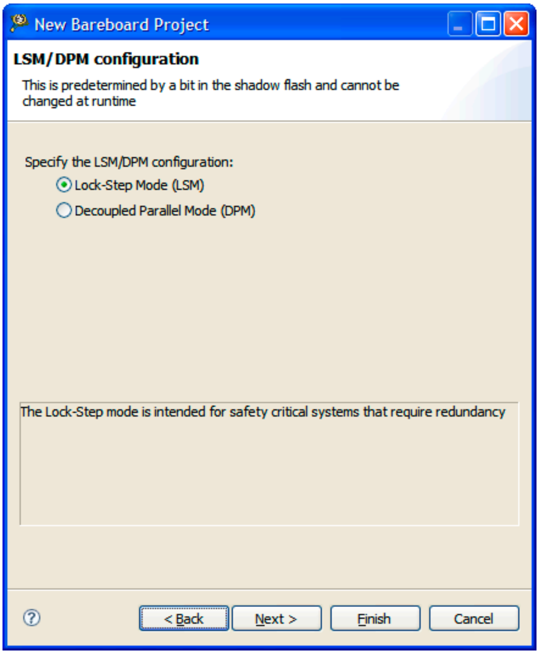
Connect to P&E USB-ML-PPCNEXUS.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

NOTE The CodeWarrior debugger supports NEXUS ISTO IEEE 5001-2003 and NEXUS ISTO IEEE 5001-2010 for MPC56xx families.

The **LSM/ DPM Configuration** page appears.

Figure 2.64 New Bareboard Project Wizard — LSM/ DPM Configuration Page



LSM/DPM configuration

This is predetermined by a bit in the shadow flash and cannot be changed at runtime

Specify the LSM/DPM configuration:

● Lock-Step Mode (LSM)

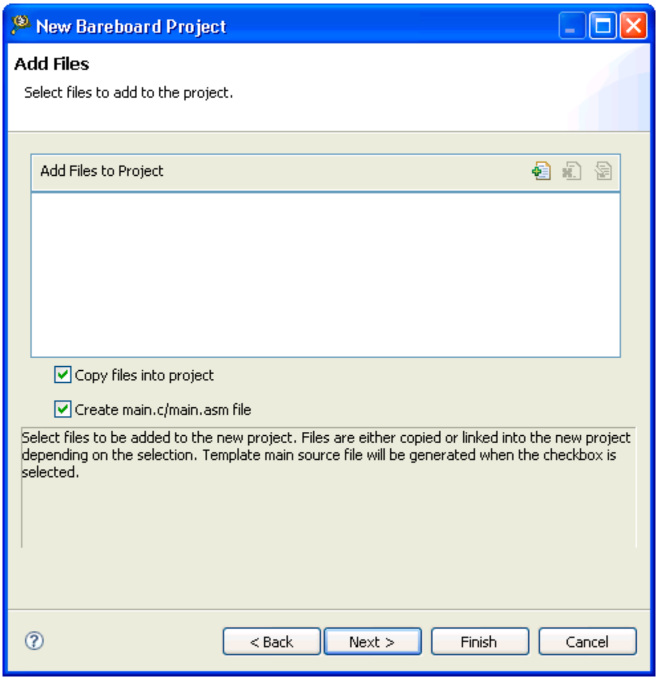
○ Decoupled Parallel Mode (DPM)

The Lock-Step mode is intended for safety critical systems that require redundancy

10. Click **Next**.

    The **Add Files** page appears.

Figure 2.65 New Bareboard Project Wizard — Add Files Page



Select files to add to the project.

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

11. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.

    b. Click **OK** to close the **Add File Path** dialog box.

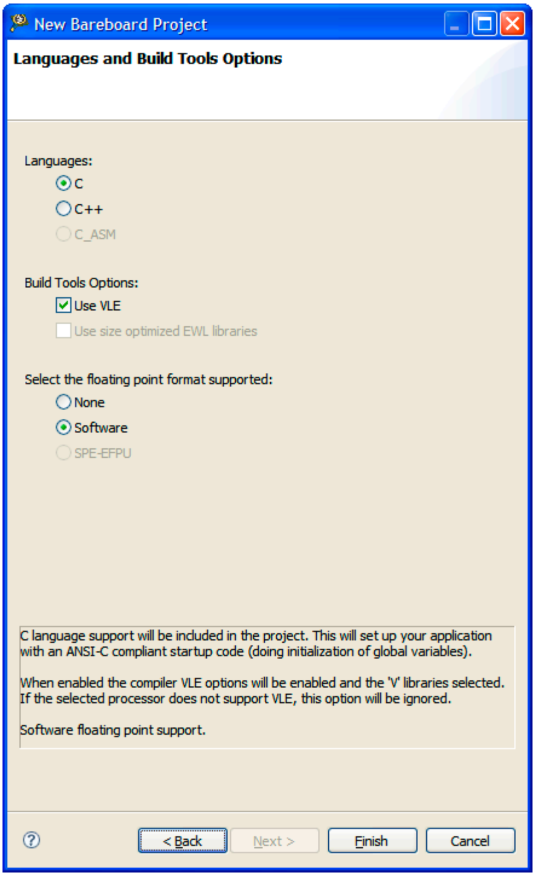
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **Languages and Build Tools Options** page appears.

Figure 2.66 New Bareboard Project Wizard — Languages and Build Tools Options Page



Languages:
● C
○ C++
○ C_ASM

Build Tools Options:
☑ Use VLE
☐ Use size optimized EWL libraries

Select the floating point format supported:
○ None
● Software
○ SPE-EFPU

C language support will be included in the project. This will set up your application with an ANSI-C compliant startup code (doing initialization of global variables).

When enabled the compiler VLE options will be enabled and the 'V' libraries selected. If the selected processor does not support VLE, this option will be ignored.

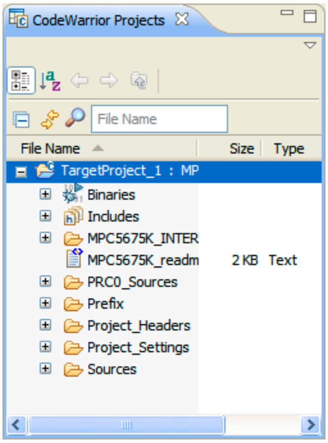
Software floating point support.

13. Select the appropriate options to enable programming language, build tools options, and floating point supports.

14. Click **Finish**.

    The wizard creates a project for the Kinetis architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.67 CodeWarrior Projects View


CodeWarrior Projects
File Name
File Name
Size
Type
TargetProject_1 : MP
Binaries
Includes
MPC5675K_INTER
MPC5675K_readm
2 KB
Text
PRC0_Sources
Prefix
Project_Headers
Project_Settings
Sources

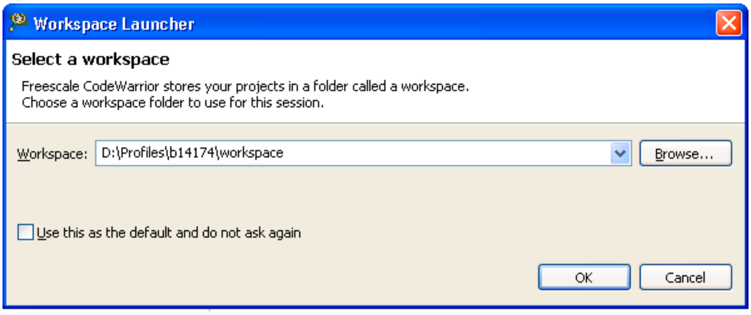
Creating Target Board Project for Kinetis

To create a project for Kinetis using the **New Bareboard Project** wizard, perform these steps:

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.68 WorkSpace Launcher Dialog Box


Workspace Launcher

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

Browse...

☐ Use this as the default and do not ask again

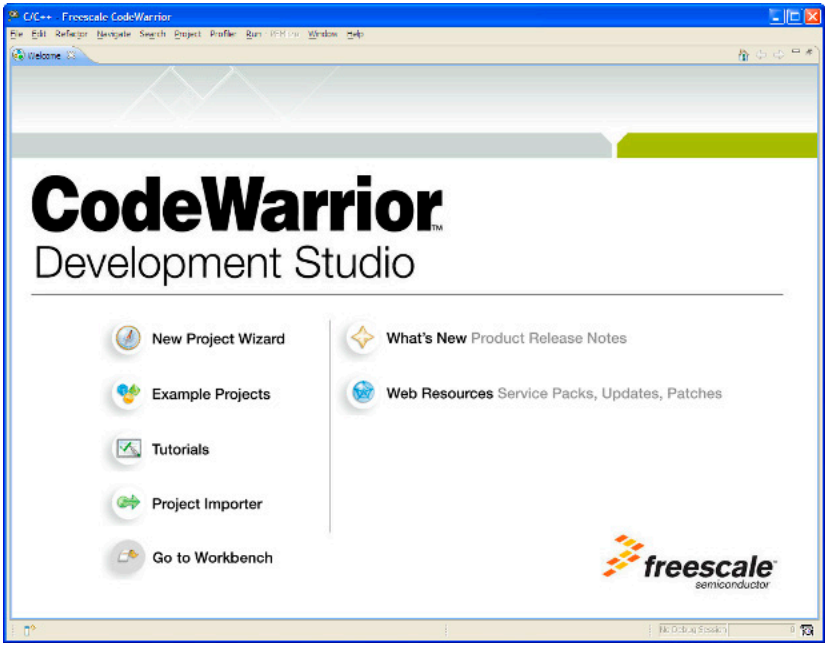
OK
Cancel

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.69 Welcome Page


C/C++ - Freescale CodeWarrior
File
Edit
Refactor
Navigate
Search
Project
Profile
Run
Window
Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale
semiconductor
No Debug Session

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

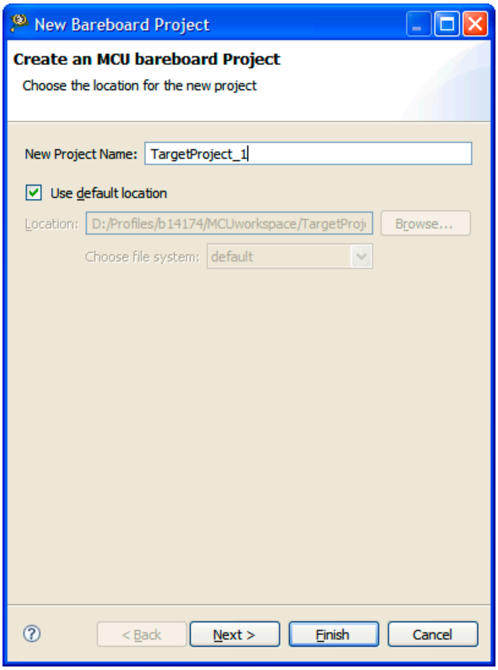
4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `TargetProject_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.70 New Bareboard Project Wizard — Create an MCU Bareboard Project Page



New Bareboard Project

Create an MCU bareboard Project

Choose the location for the new project

New Project Name: TargetProject_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/TargetProj

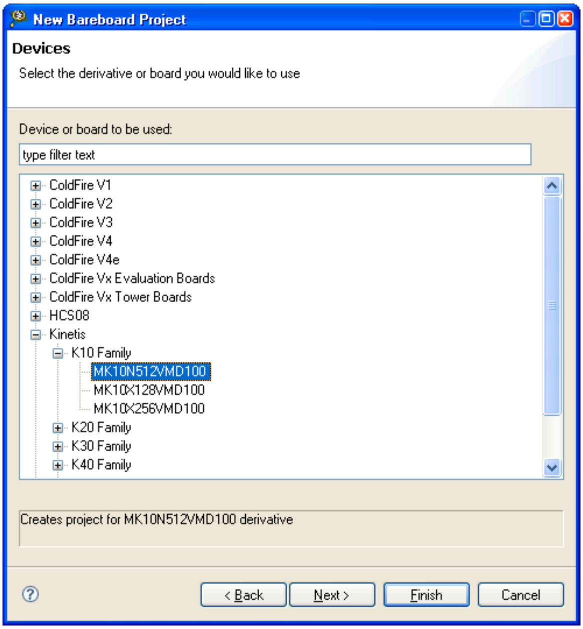
Choose file system: default

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select **Kinetis > K10 Family > MK10N512VMD100**.

Figure 2.71 New Bareboard Project Wizard — Devices

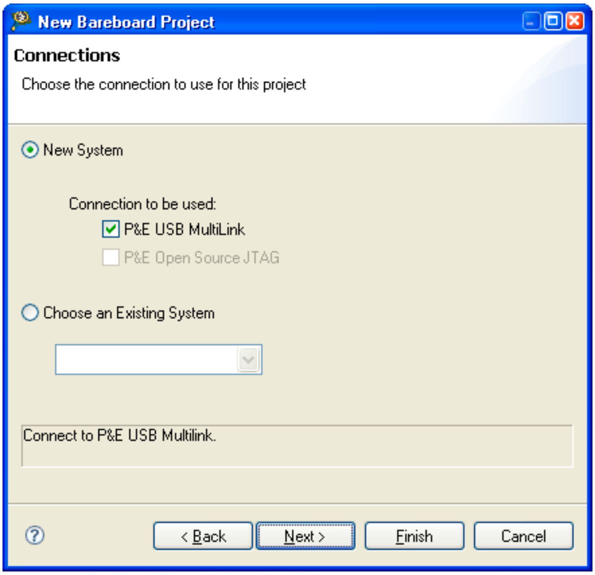

New Bareboard Project
Devices
Select the derivative or board you would like to use
Device or board to be used:
type filter text
ColdFire V1
ColdFire V2
ColdFire V3
ColdFire V4
ColdFire V4e
ColdFire Vx Evaluation Boards
ColdFire Vx Tower Boards
HCS08
Kinetis
K10 Family
MK10N512VMD100
MK10X128VMD100
MK10X256VMD100
K20 Family
K30 Family
K40 Family
Creates project for MK10N512VMD100 derivative
< Back
Next >
Finish
Cancel

8. Click **Next**.

   The **Connections** page appears.

9. Check the appropriate connection. For example, check the **P&E USB MultiLink** checkbox.

Figure 2.72 New Bareboard Project Wizard — Connections Page

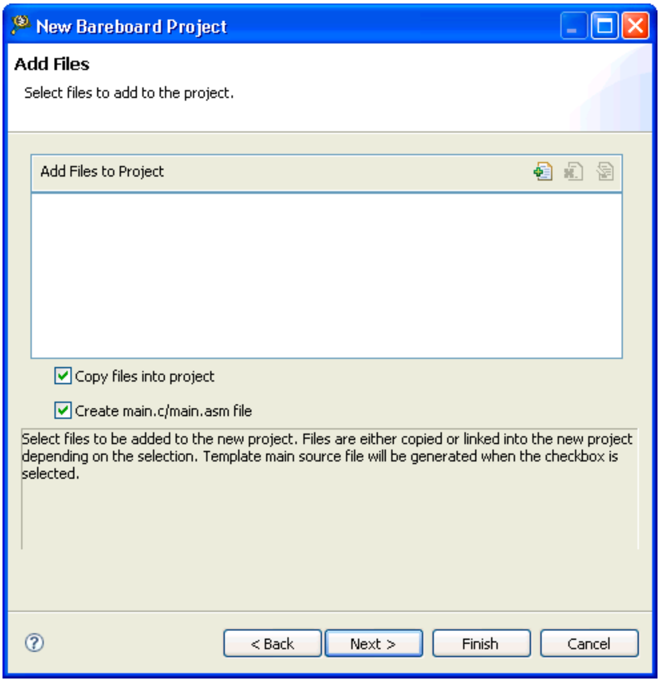


New Bareboard Project
Connections
Choose the connection to use for this project
● New System
Connection to be used:
☑ P&E USB MultiLink
☐ P&E Open Source JTAG
○ Choose an Existing System
Connect to P&E USB Multilink.
< Back
Next >
Finish
Cancel

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

The **Add Files** page appears.

Figure 2.73 New Bareboard Project Wizard — Add Files Page



Add Files

Select files to add to the project.

Add Files to Project

☑ Copy files into project

☑ Create main.c/main.asm file

Select files to be added to the new project. Files are either copied or linked into the new project depending on the selection. Template main source file will be generated when the checkbox is selected.

10. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.

    b. Click **OK** to close the **Add File Path** dialog box.

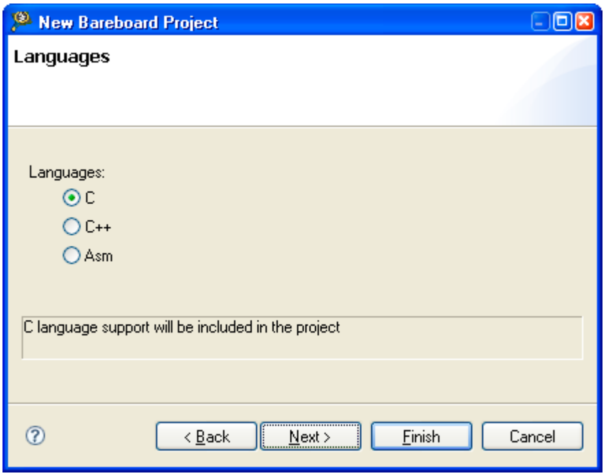
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

    d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

11. Click **Next**.

    The **Languages** page appears.

Figure 2.74 New Bareboard Project Wizard — Languages Page


New Bareboard Project
Languages
Languages:
● C
○ C++
○ Asm
< Back
Next >
Finish
Cancel

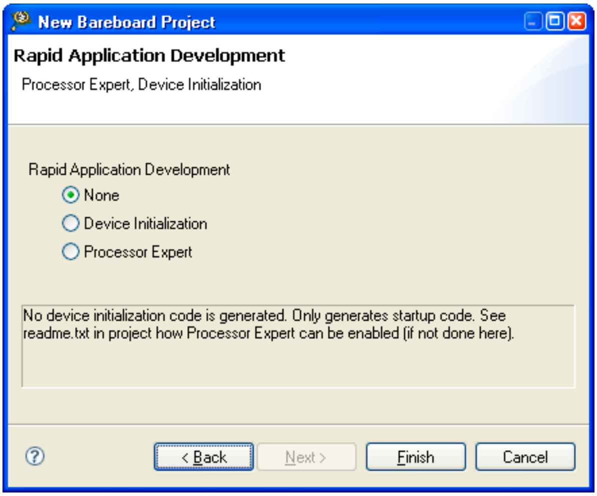
C language support will be included in the project

12. Select the appropriate options to enable C++, porting processor macro, console, and floating point supports.

13. Click **Next**.

    The **Rapid Application Development** page appears.

Figure 2.75 New Bareboard Project Wizard — Rapid Application Development Page


New Bareboard Project
Rapid Application Development
Processor Expert, Device Initialization
Rapid Application Development
● None
○ Device Initialization
○ Processor Expert
< Back
Next >
Finish
Cancel

No device initialization code is generated. Only generates startup code. See readme.txt in project how Processor Expert can be enabled (if not done here).

14. Select the appropriate option to support rapid application development.

    - **None** — Select to generate only startup code.

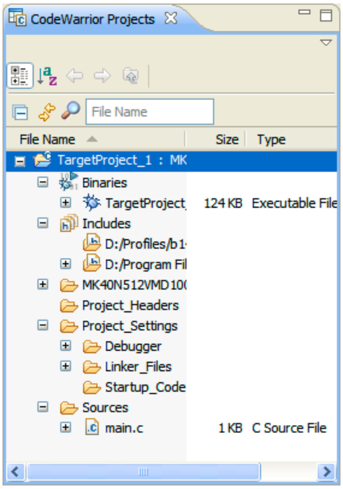
- **Device Initialization** — Select to generate the initialization code for on-chip peripherals, interrupt vector table, and template for interrupt vector service routines.
- **Processor Expert** — Select to generate the device initialization code, including low-level drivers.

NOTE If you select the **Processor Expert** option, clicking the Next button will display the **Processor Expert MCU Pin Variants and Configuration** page. Here you can select the required microcontroller variant and its configuration.

15. Click **Finish**.

    The wizard creates a project for the Kinetis architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 2.76 CodeWarrior Projects View


CodeWarrior Projects
File Name
File Name
Size
Type
TargetProject_1 : MK
Binaries
TargetProject
124 KB
Executable File
Includes
D:/Profiles/b1
D:/Program Fil
MK40N512VMD10
Project_Headers
Project_Settings
Debugger
Linker_Files
Startup_Code
Sources
main.c
1 KB
C Source File

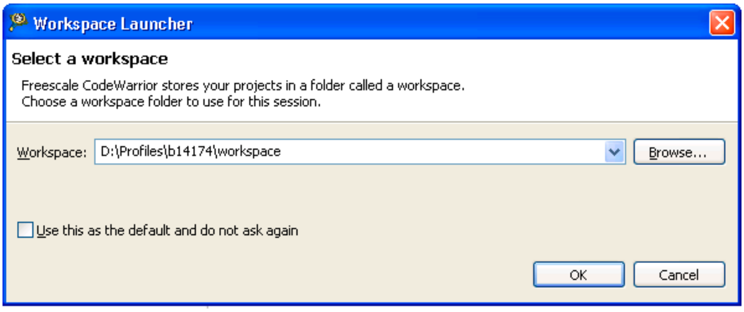
Creating Linux/uClinux Application Project

To create a Linux/uClinux Application Project using the **New Linux/uClinux Application Project** wizard, perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 2.77 WorkSpace Launcher Dialog Box



Workspace Launcher

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

☐ Use this as the default and do not ask again

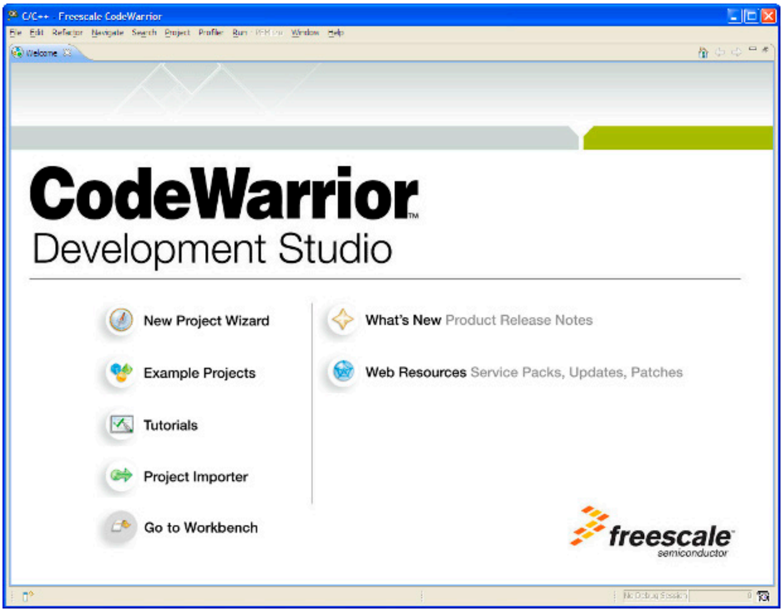
Browse...
OK
Cancel

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 2.78 Welcome Page



C/C++ - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run Window Help
Welcome
CodeWarrior
Development Studio
New Project Wizard
Example Projects
Tutorials
Project Importer
Go to Workbench
What's New Product Release Notes
Web Resources Service Packs, Updates, Patches
freescale semiconductor

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

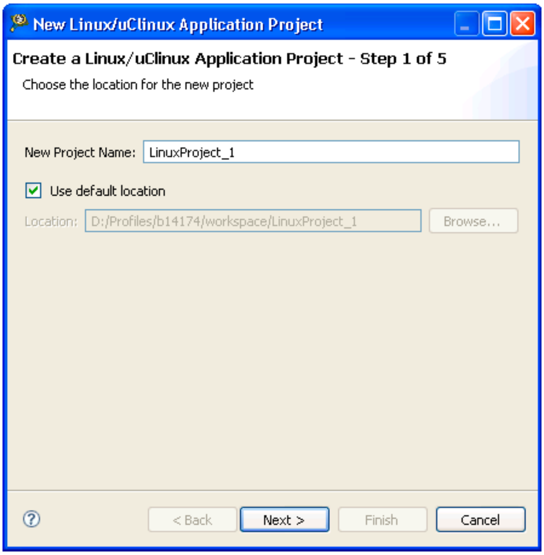
4. Select **File > New > Linux/uClinux Application Project**, from the IDE menu bar.

   The **Create a Linux/uClinux Application Project** page of the **New Linux/uClinux Application Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `LinuxProject_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 2.79 Create a Linux/uClinux Application Project Page



Create a Linux/uClinux Application Project - Step 1 of 5

Choose the location for the new project

New Project Name: LinuxProject_1

☑ Use default location

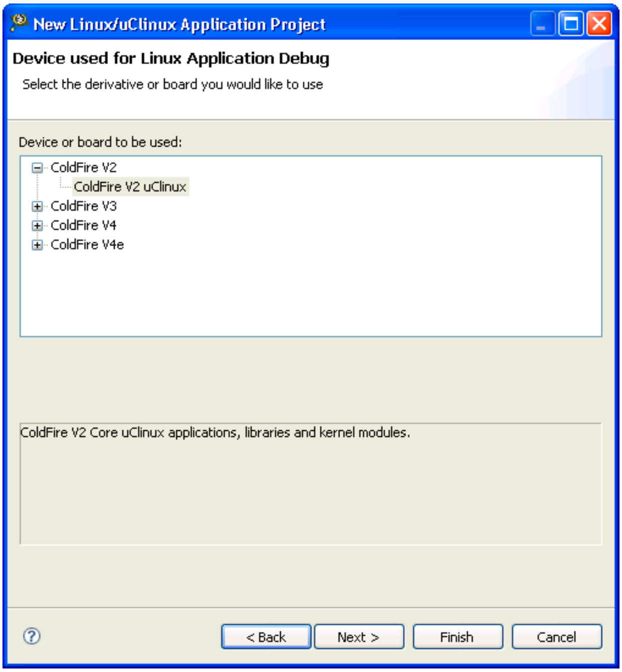
Location: D:/Profiles/b14174/workspace/LinuxProject_1

6. Click **Next**.

   The **Device used for Linux Application Debug** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select `ColdFire V2 > ColdFire V2 uClinux`.

Figure 2.80 Device used for Linux Application Debug Page

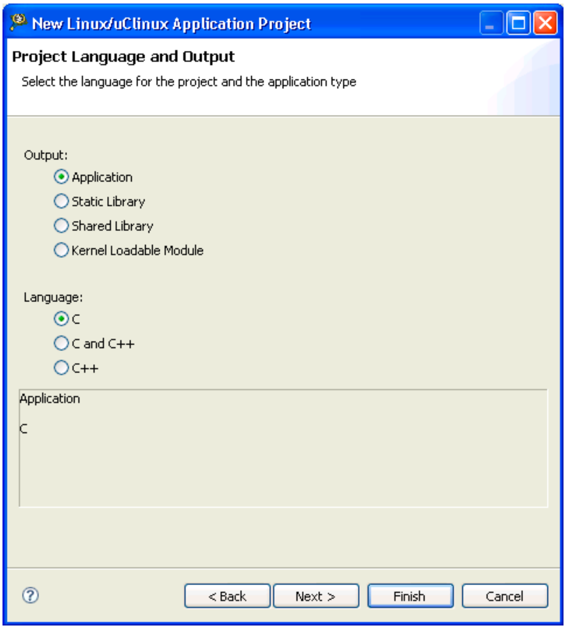

New Linux/uClinux Application Project
Device used for Linux Application Debug
Select the derivative or board you would like to use
Device or board to be used:
ColdFire V2
ColdFire V2 uClinux
ColdFire V3
ColdFire V4
ColdFire V4e
< Back
Next >
Finish
Cancel

ColdFire V2 Core uClinux applications, libraries and kernel modules.

8. Click **Next**.

   The **Project Language and Output** page (Figure 2.89) appears.

Figure 2.81 Project Language and Output Page

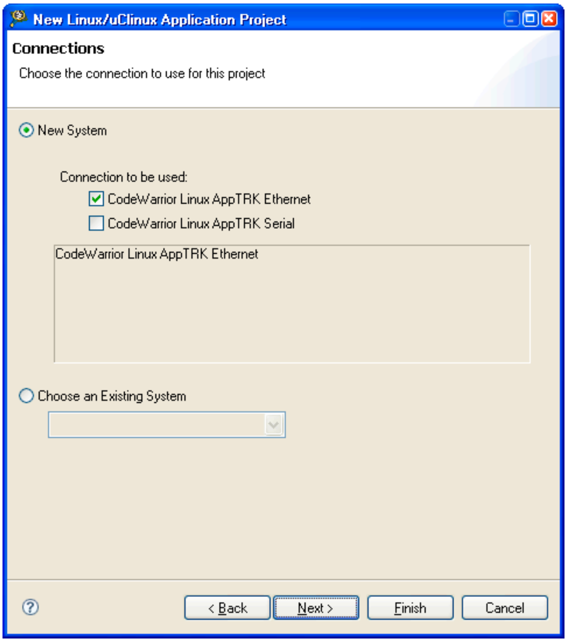

New Linux/uClinux Application Project
Project Language and Output
Select the language for the project and the application type
Output:
● Application
○ Static Library
○ Shared Library
○ Kernel Loadable Module
Language:
● C
○ C and C++
○ C++
Application
C
< Back
Next >
Finish
Cancel

9. Select the output type and the programming language you want to use for this project. For example, select **Application** and **C and C++**.

10. Click **Next**.

    The **Connections** page appears.

Figure 2.82 Connections Page



New Linux/uClinux Application Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☑ CodeWarrior Linux AppTRK Ethernet

☐ CodeWarrior Linux AppTRK Serial

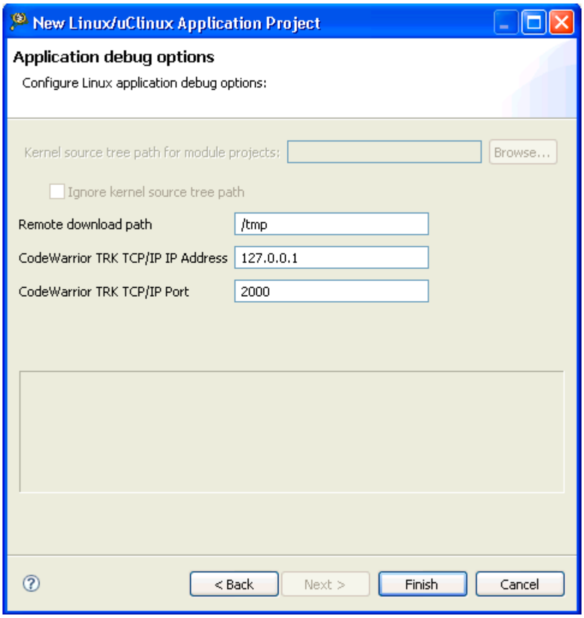
CodeWarrior Linux AppTRK Ethernet

○ Choose an Existing System

11. Check the appropriate connection.
12. Click **Next**.

    The **Application debug options** page appears.

Figure 2.83 Application Debug Options Page



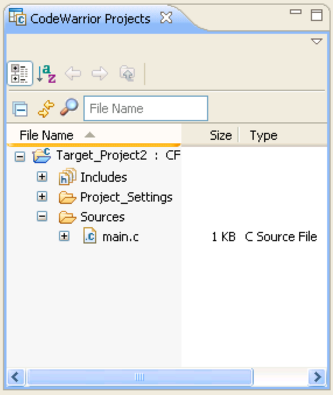
New Linux/uClinux Application Project
Application debug options
Configure Linux application debug options:
Kernel source tree path for module projects:
Browse...
☐ Ignore kernel source tree path

Remote download path: /tmp
CodeWarrior TRK TCP/IP IP Address: 127.0.0.1
CodeWarrior TRK TCP/IP Port: 2000

< Back
Next >
Finish
Cancel

13. From the list, select the method with which you want the IDE to connect to the target system.
14. In the **Remote download path** text box, specify the path. By default, it is `/tmp`.
15. In the **CodeWarrior TRK TCP/IP Address** and **CodeWarrior TCP/IP** text boxes, enter the IP address and listening port of the target system. By default, the value of the TCP/IP address is `127.0.0.1` and the port number is `2000`.
16. Click **Finish**.
    - The wizard closes. The IDE generates a new project according to your specifications. The **CodeWarrior Projects** view (Figure 2.84) appears.

Figure 2.84 CodeWarrior Projects View


CodeWarrior Projects
File Name
File Name
Size
Type
Target_Project2 : CF
Includes
Project_Settings
Sources
main.c
1 KB
C Source File

Building Projects

CodeWarrior IDE supports two modes of building projects:

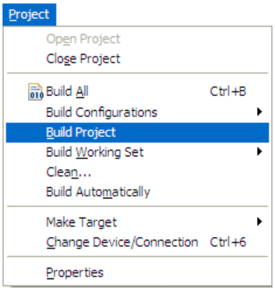
- Manual-Build Mode
- Auto-Build Mode

Manual-Build Mode

In large workspaces, building the entire workspace can take a long time if you make changes with a significant impact on dependent projects. Often there are only a few projects that really matter to you at a given time.

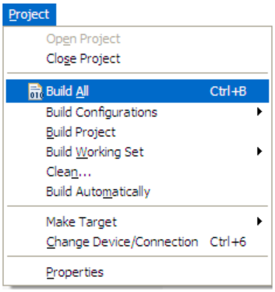
To build only the selected projects, and any prerequisite projects that need to be built in order to correctly build the selected projects, select **Project > Build Project** from the CodeWarrior IDE menu bar (Figure 2.85).

Figure 2.85 Project Menu — Build Project


Project
Open Project
Close Project
Build All Ctrl+B
Build Configurations
Build Project
Build Working Set
Clean...
Build Automatically
Make Target
Change Device/Connection Ctrl+6
Properties

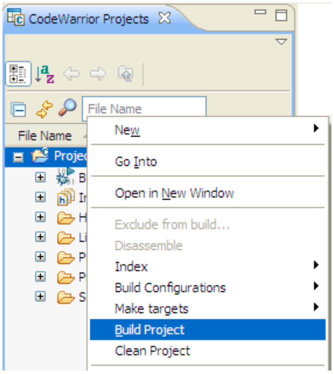
Alternatively, select **Project > Build All** (Figure 2.86).

Figure 2.86 Project Menu — Build All


Project
Open Project
Close Project
Build All Ctrl+B
Build Configurations
Build Project
Build Working Set
Clean...
Build Automatically
Make Target
Change Device/Connection Ctrl+6
Properties

Alternatively, right-click on the simulator project in the **CodeWarrior Projects** view. A context menu appears. From the context menu, select **Build Project**. The IDE builds the new project (Figure 2.87).

Figure 2.87 Context Menu — Build Project

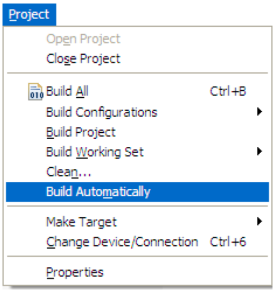

CodeWarrior Projects
File Name
File Name
Projec
New
Go Into
Open in New Window
Exclude from build...
Disassemble
Index
Build Configurations
Make targets
Build Project
Clean Project

Auto-Build Mode

The CodeWarrior IDE takes care of compiling source files automatically. Builds occur automatically in the background every time you change files in the workspace (for example, saving a file in an editor), if auto-build is enabled.

To automatically build all the projects in a workspace, select **Project > Build Automatically** from the CodeWarrior IDE menu bar (Figure 2.88).

Figure 2.88 Project Menu — Build Automatically


Project
Open Project
Close Project
Build All Ctrl+B
Build Configurations
Build Project
Build Working Set
Clean...
Build Automatically
Make Target
Change Device/Connection Ctrl+6
Properties

Debugging Projects

When you use the **New Bareboard Project** wizard to create a new project, the wizard sets the debugger settings of the project's launch configurations to default values. You can change these default values based on your requirements.

To debug a project, perform these steps.

1. Launch the IDE.
2. From the main menu bar of the IDE, select **Run > Debug Configurations**. The IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

   The **Debug Configurations** dialog box appears. The left side of this dialog box has a list of debug configurations that apply to the current application.

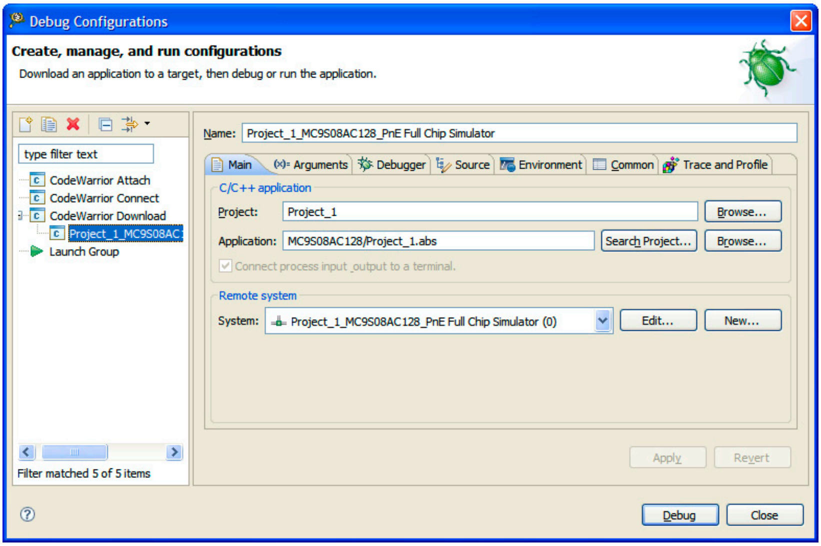
NOTE For more information on how to use the debugger, refer to the *Freescale Eclipse Extensions Guid*e and the Working with Debugger chapter of this manual.

3. Expand the **CodeWarrior Download** configuration.

4. From the expanded list, select the debug configuration that you want to modify.

   Figure 2.89 displays the **Debug Configurations** dialog box with the settings for the debug configuration you selected.

Figure 2.89 Debug Configurations Dialog Box


Debug Configurations
type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Project_1_MC9S08AC
Launch Group
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
C/C++ application
Remote system
Browse...
Search Project...
Browse...
Edit...
New...
Apply
Revert
Debug
Close

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Project_1_MC9S08AC128_PnE Full Chip Simulator

Project: Project_1

Application: MC9S08AC128/Project_1.abs

☑ Connect process input_output to a terminal.

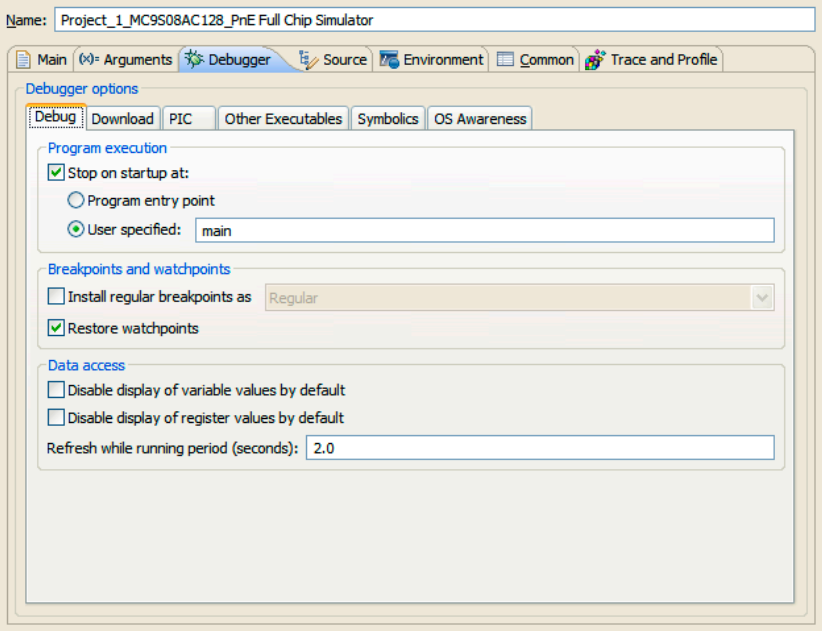
System: Project_1_MC9S08AC128_PnE Full Chip Simulator (0)

Filter matched 5 of 5 items

5. Click the **Debugger** tab.

   The **Debugger** page appears in the area beneath the tabs.

Figure 2.90 Debug Configurations Dialog Box — Debugger Page


Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
Debugger options
Debug
Download
PIC
Other Executables
Symbolics
OS Awareness

Name: Project_1_MC9S08AC128_PnE Full Chip Simulator

Program execution

☑ Stop on startup at:

○ Program entry point

● User specified: main

Breakpoints and watchpoints

☐ Install regular breakpoints as Regular

☑ Restore watchpoints

Data access

☐ Disable display of variable values by default

☐ Disable display of register values by default

Refresh while running period (seconds): 2.0

6. Change the settings on this page as per your requirements.

NOTE For more information on debugger, refer to the chapter Working with Debugger.

7. Click **Apply** to save the new settings.
8. Click **Debug** to start the debugging session.

   You just finished starting a debugging session and attaching the debugger to a process.

NOTE You can click **Revert** to undo any of the unsaved changes. The IDE restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables **Revert** until you make new pending changes.

Deleting Projects

To delete a project, follow these steps.

1. Select the project you want to delete in the **CodeWarrior Projects** view.
2. Select **Edit** > **Delete**.

   The **Confirm Project Delete** dialog box appears.

NOTE Alternatively, you can also select **Delete** from the context menu when you right-click on the project.

3. Select the **Also delete contents under** ***<filepath>*** option if you want to delete the contents of the selected project. Else, select the **Do not delete contents** option.

NOTE You will not be able to restore your project using “Undo”, if you select the **Also delete contents under** ***<filepath>*** option.

4. Click **Yes**.

   The project is removed from the **CodeWarrior Projects** view.

Importing Classic CodeWarrior Projects

The CodeWarrior Classic Project Importer feature in Eclipse helps automate the conversion of a legacy C/C++ CodeWarrior 5.x project to an Eclipse CDT project.

This feature lets you:

- select the classic CodeWarrior project,
- set targets to import,
- configure source trees and shielded folders,
- edit access paths for each target,
- list files that are not found in the previous settings,
- specify the new Eclipse project name and location,
- list warning or errors in the conversion process, and
- open the newly created Eclipse project.

NOTE For more information on importing classic CodeWarrior projects to Eclipse IDE, refer to the *Freescale Eclipse Extensions Guide* and *CodeWarrior Project Importer Quick Start*.

NOTE For information on importing connection-specific Microcontroller projects, refer to the Tutorials — Importing Connection-Specific Projects topic.

Tutorials — Importing Connection-Specific Projects

This topic consists of tutorials that demonstrate how to import connection-specific classic Microcontrollers projects to Eclipse.

The tutorials include:

- Tutorial A: Porting Classic HCS08 Project
- Tutorial B: Porting Classic RS08 Project
- Tutorial C: Porting Classic ColdFire V1 Project
- Tutorial D: Porting Classic ColdFire V2/3/4 Project
- Tutorial E: Porting Classic Power Architecture Project

Tutorial A: Porting Classic HCS08 Project

The goal of this tutorial is to import a classic HCS08 project to Eclipse.

NOTE Before starting the import process ensure that the CodeWarrior HCS08 project you want to import has all of its files, such as the source, linker command, and settings file.

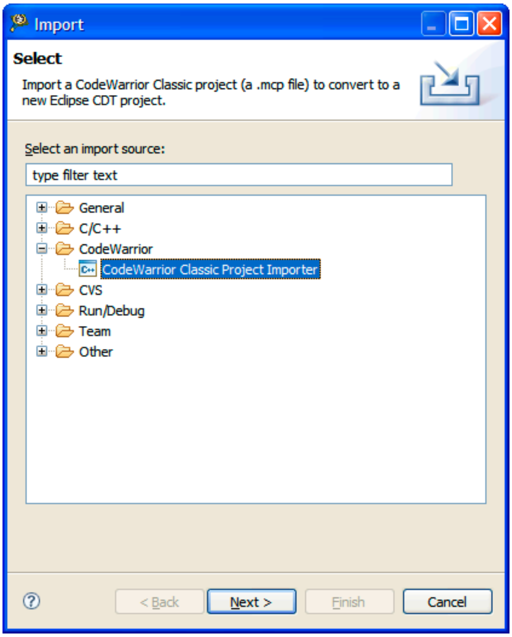
To port a classic HCS08 project, perform these steps.

1. Select **File > Import** from the **Workbench** menu bar.

   The **Import** dialog box appears.

2. Expand the **CodeWarrior** tree control and select **CodeWarrior Classic Project Importer**.

Figure 2.91 Import Dialog Box



Import
Select
Import a CodeWarrior Classic project (a .mcp file) to convert to a new Eclipse CDT project.
Select an import source:
type filter text
General
C/C++
CodeWarrior
CodeWarrior Classic Project Importer
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

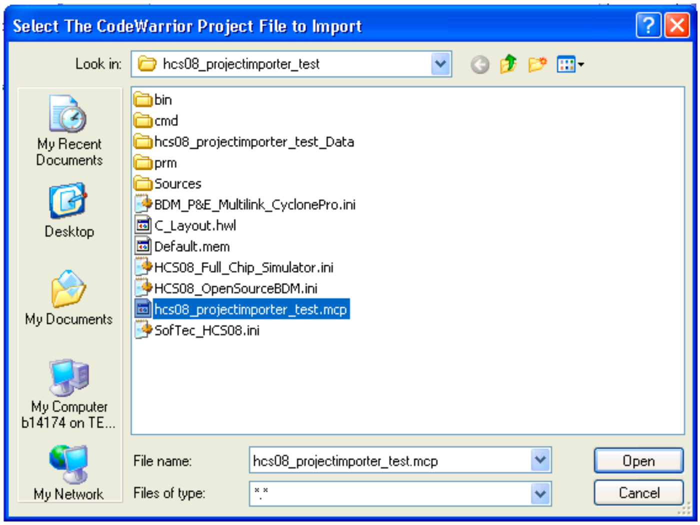
3. Click **Next**.

   The first page of the **CodeWarrior Project Importer** wizard appears.

4. Enter the path and name of the classic CodeWarrior project file to import in the **Project file** text box. Alternatively, click **Browse** and use the **Select The CodeWarrior Project File to Import** dialog box to select the project file to import. In this case, assume that the classic CodeWarrior project filename is `hcs08_projectimporter_test.mcp`.

TIP The project file has an extension of `.mcp`. Select the `.mcp` file.

Figure 2.92 Select The CodeWarrior Project File to Import Dialog Box


Select The CodeWarrior Project File to Import

Look in: hcs08_projectimporter_test

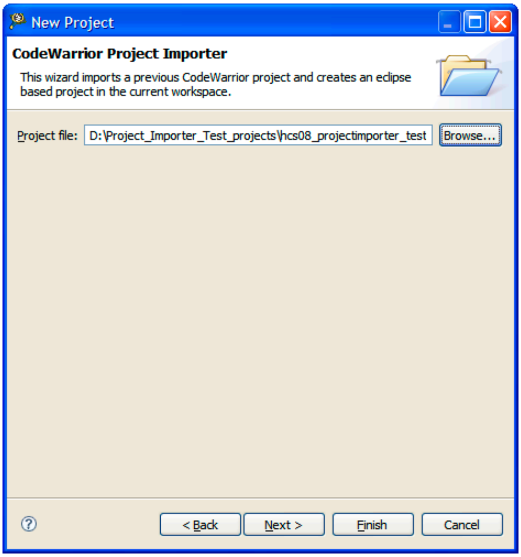
File name: hcs08_projectimporter_test.mcp

Files of type: *.*

My Recent Documents
Desktop
My Documents
My Computer b14174 on TE...
My Network
bin
cmd
hcs08_projectimporter_test_Data
prm
Sources
BDM_P&E_Multilink_CyclonePro.ini
C_Layout.hwl
Default.mem
HCS08_Full_Chip_Simulator.ini
HCS08_OpenSourceBDM.ini
hcs08_projectimporter_test.mcp
SofTec_HCS08.ini
Open
Cancel

The path of the project file to import appears in the **Project file** text box.

Figure 2.93 Path of CodeWarrior Project File to Import


New Project

CodeWarrior Project Importer

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

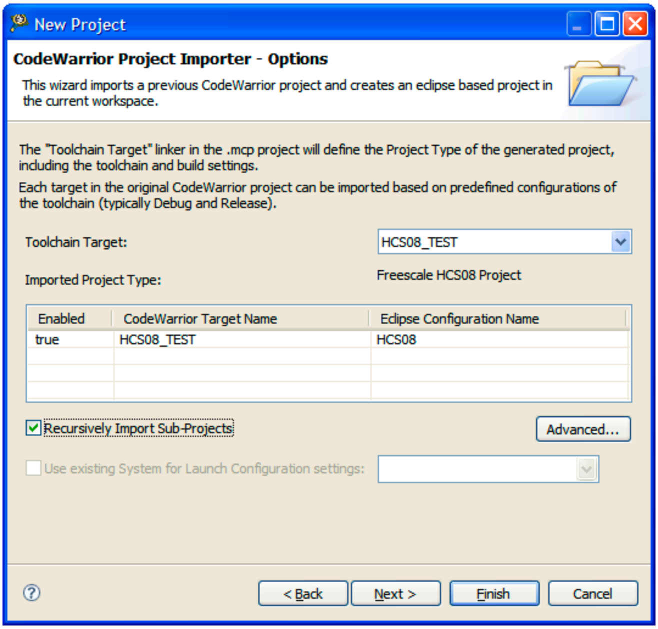
Project file: D:\Project_Importer_Test_projects\hcs08_projectimporter_test

Browse...
< Back
Next >
Finish
Cancel

5. Click **Next**.

   The **Options** page of the **CodeWarrior Classic Project Importer** wizard appears.

Figure 2.94 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: HCS08_TEST

Imported Project Type: Freescale HCS08 Project

Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	HCS08_TEST	HCS08

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

6. Select the build target that uses the HCS08 toolchain you want the generated Eclipse project to use, from the **Toolchain Target** list box.

NOTE The toolchain target linker in the classic project defines the project type of the generated Eclipse project, including the toolchain and build settings.

The build targets table displays all the targets discovered in the project file and is used to generate equivalent Eclipse build configurations.

7. You can import each build target in the classic CodeWarrior project based upon predefined configurations of the toolchain. For example: The HCS08 project in the example lists these values:
   - **Imported Project Type** = *Freescale HCS08 Project*
   - **Enabled** = *true*
   - **CodeWarrior Target Name** = <*Toolchain Target*>
   - **Eclipse Configuration Name** = *HCS08*

TIP To disable the generation of specific configurations, click a row in the build target table. In the **Edit Table Values** dialog box set **Enabled** to *false* and click **OK**.

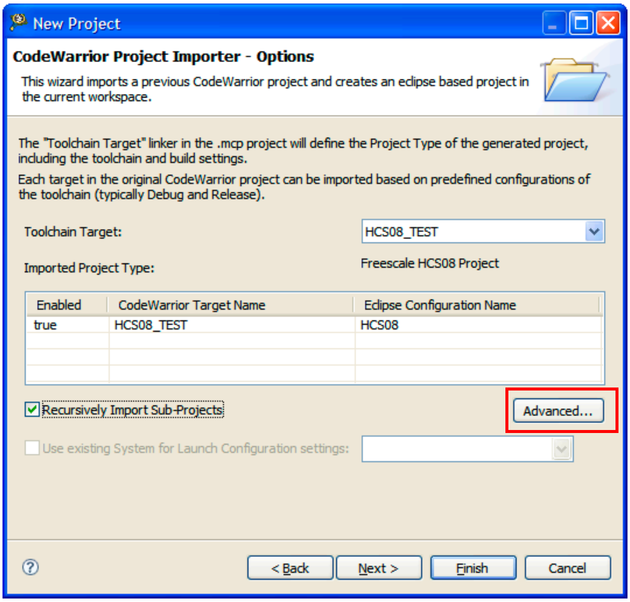
8. If you want to import sub-projects included in the classic CodeWarrior project, check the **Recursively Import Sub-Projects** checkbox.

   The **CodeWarrior Classic Project Importer** wizard imports the sub-projects with the main project.

NOTE The **CodeWarrior Classic Project Importer** wizard will copy only those files that are displayed in the **CodeWarrior Projects** view. The wizard will not import a file if it is not displayed in the view or does not include project information.

9. Click **Advanced**.

Figure 2.95 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: HCS08_TEST

Imported Project Type: Freescale HCS08 Project

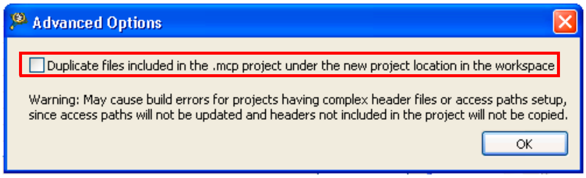
Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	HCS08_TEST	HCS08

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

The **Advanced Options** dialog box appears.

Figure 2.96 Advanced Options



☐ Duplicate files included in the .mcp project under the new project location in the workspace

Warning: May cause build errors for projects having complex header files or access paths setup, since access paths will not be updated and headers not included in the project will not be copied.

10. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox.

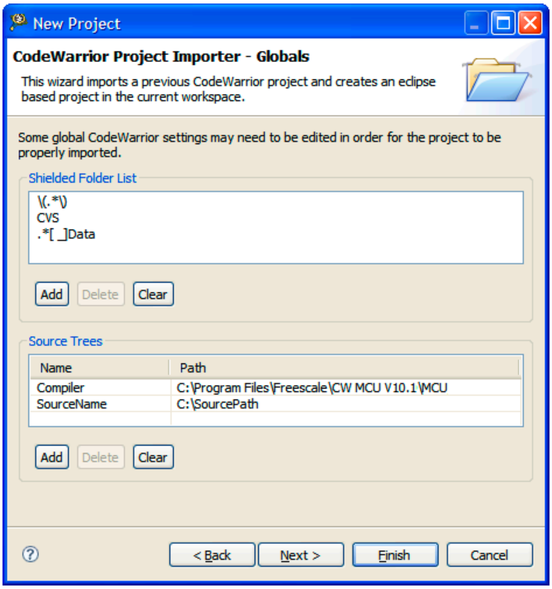
WARNING! Checking the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox may cause build errors. To correct this you may need to add the libraries to the **Libraries Files** option in **Build Properties > Linker** options. This will enable the project to compile and link successfully. For more information on the build properties, refer to the chapter Build Properties for Bareboard Projects.

---

11. Click **OK** to close the dialog box.
12. Click **Next**.

    The **Globals** page of the **CodeWarrior Classic Project Importer** wizard appears. This page lets you edit global settings that can effect how the project's build options are imported.

Figure 2.97 CodeWarrior Project Importer — Globals Page



CodeWarrior Project Importer - Globals

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Some global CodeWarrior settings may need to be edited in order for the project to be properly imported.

Shielded Folder List

```
\(.*\)
CVS
.*[ _]Data
```

Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

Table 2.19 lists the options on the **CodeWarrior Classic Project Importer — Globals** page.

Table 2.19 CodeWarrior Classic Project Importer — Globals Page Options

Options	Description
Shielded Folder List	Lists the concealed contents of folders from the IDE's search operations during a build. This was done by placing special characters in the directory name. For example, sample code was concealed in a (CodeWarrior Examples) folder. The **Shielded Folder List** table lists these options: • \(.*\) • CVS • .*[_]Data You use the **Add, Delete**, and **Clear** buttons to modify the information in this list. Table 2.20 lists these buttons with their descriptions.
Source Trees	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The **{Project}** source tree is defined automatically. The **Sources Trees** table lists these options: • Name — Lists the source name. For example: Compiler. • Compiler — Lists the path source name. For example: `<CW Install>/MCU`. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.20 lists these buttons with their descriptions.

Table 2.20 CodeWarrior Classic Project Importer — Globals Page Buttons

Button	Description
Add	Add a new entry to the list.
Delete	Deletes the selected item.
Clear	Clears the entire list.

13. To add a new expression to the **Shielded Folder List** table, perform these steps.

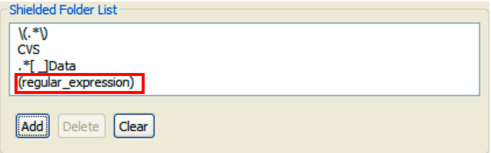
    a. Click **Add**.

    (regular_expression) appears in the shielded folder list.

b. Double-click *(regular_expression)* and type the required expression.

The new expression appears in shielded folder list.

Figure 2.98 Shielded Folder List



```
\(.*\)
CVS
.*[ _]Data
(regular_expression)
```

14. To delete an existing expression from the **Shielded Folder List** table, select the expression and click **Delete**.

    The selected expression is deleted from the shielded folder list.

15. To remove all the existing expressions from the **Shielded Folder List** table, click **Clear**.

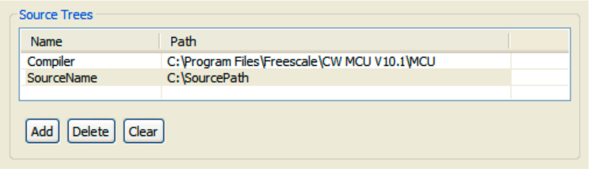
    All the expressions are deleted from the shielded folder list.

16. To add a new source to the **Source Trees** table, perform these steps.

    a. Click **Add**.

    SourceName appears in the source trees list.

Figure 2.99 Source Trees

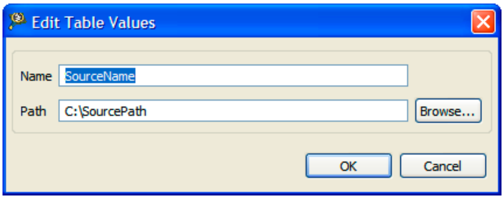


Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

b. Double-click *SourceName*.

The **Edit Table Values** dialog box appears.

Figure 2.100 Edit Table Values



Name: SourceName
Path: C:\SourcePath

c. In the **Name** text box, enter the source name.

d. In the **Path** text box, enter the path of the new source. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

e. Click **OK**.

The new source appears in **Source Trees** list.

17. Click **Next**.

The **CodeWarrior Project Importer - Access Paths** page appears.

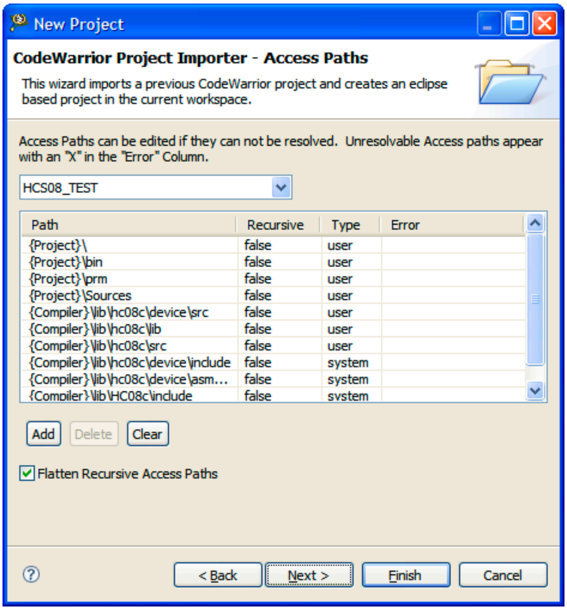
NOTE Access paths are directory paths the CodeWarrior tools use to search for libraries, runtime support files, and other object files.

Table 2.21 lists the options on the **CodeWarrior Project Importer — Access Paths** page.

Table 2.21 CodeWarrior Project Importer — Access Paths Page

Option	Description
Build Target List Box	Lists the build target whose access paths you want to modify. For example: `HCS08_TEST`
Access Path Table	Lists the access paths used by the build target selected in the **Build Target** list box. Each row lists: • **Path** — Directory path. For example: `{Compiler}\lib\HC08c\include.` • **Recursive** — Whether the path is searched recursively. For example: `false` or `true.` • **Type** — Specify of path to be searched. For example: `user` or `system.` • **Error** — Unresolved access paths, marked as “X”, if any. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.22 lists these buttons with their descriptions.
Flatten Recursive Access Paths Checkbox	Check to enable the **CodeWarrior Classic Project Importer** wizard automatically generate separate include paths for each subdirectory that is a part of the recursive path. This option is set by default as most compilers do not support recursive include paths passed by the command line.

Figure 2.101 CodeWarrior Project Importer — Access Paths Page



CodeWarrior Project Importer - Access Paths

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Access Paths can be edited if they can not be resolved. Unresolvable Access paths appear with an "X" in the "Error" Column.

HCS08_TEST

Path	Recursive	Type	Error
{Project}\	false	user	
{Project}\bin	false	user	
{Project}\prm	false	user	
{Project}\Sources	false	user	
{Compiler}\lib\hc08c\device\src	false	user	
{Compiler}\lib\hc08c\lib	false	user	
{Compiler}\lib\hc08c\src	false	user	
{Compiler}\lib\hc08c\device\include	false	system	
{Compiler}\lib\hc08c\device\asm...	false	system	
{Compiler}\lib\HC08c\include	false	system	

☑ Flatten Recursive Access Paths

Table 2.22 CodeWarrior Project Importer — Access Paths Page Buttons

Button	Description
Add	Add a new directory path to the list.
Delete	Deletes the selected directory path from the list.
Clear	Clears the entire list.

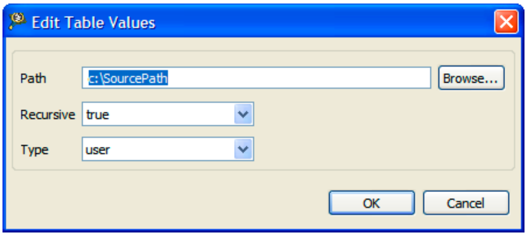
18. To add a new directory path to list, perform these steps.
    a. Click **Add**.

       (C:\SourcePath) appears in the **Path** list.

    b. Click *(C:\SourcePath)*.

       The **Edit Table Values** dialog box appears.

Figure 2.102 Edit Table Values



Path: c:\SourcePath
Recursive: true
Type: user

c. In the **Path** text box, enter the directory path. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

d. From the **Recursive** list box, select `false` or `true`.

e. From the **Type** list box, select `user` or `system`.

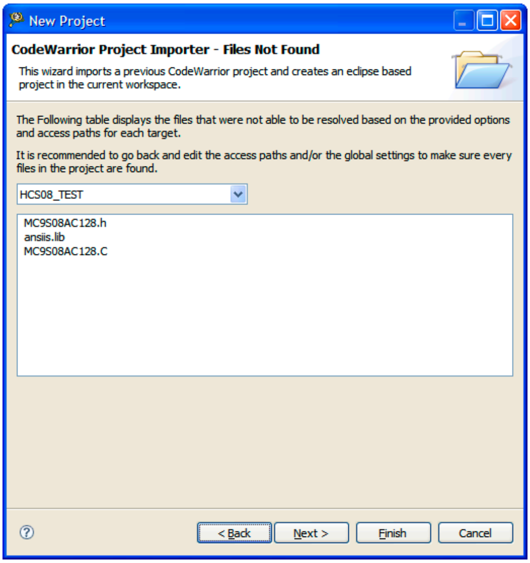
f. Click **OK**.

The new access path appears in table.

19. Click **Next**.

The **CodeWarrior Project Importer — Files Not Found** page appears. This page displays the project files that the wizard could not locate. You can use the Build Target list box to select another build target and view the missing files.

Figure 2.103 CodeWarrior Project Importer — Files Not Found Page



CodeWarrior Project Importer - Files Not Found

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

HCS08_TEST

MC9S08AC128.h
ansiis.lib
MC9S08AC128.C

20. To locate the missing files, perform these steps.

a. Click **Back** to adjust the settings in the **Globals and Access Paths** pages so that the wizard can locate the missing files.

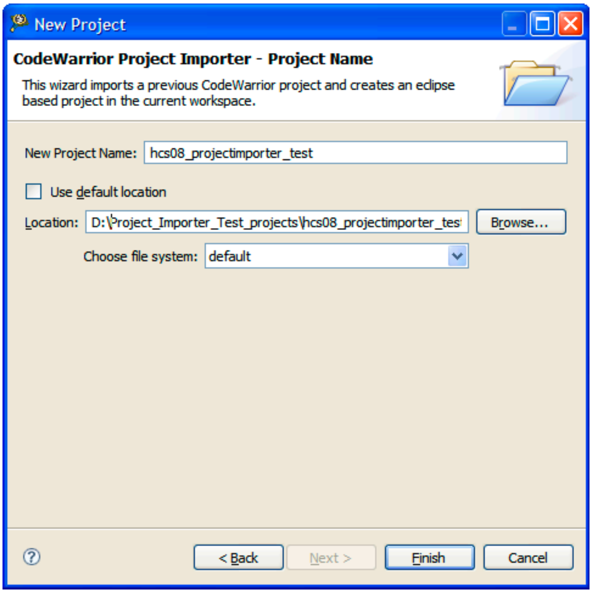
b. Repeat till you narrow down the number of missing files.

NOTE Some old files do not work with the <*target*> implementation, thus there might be some files missing.

21. Click **Next**.

The **CodeWarrior Project Importer - Project Name** page appears. This page lets you specify the name and select a location for the newly imported project.

Figure 2.104 CodeWarrior Project Importer — Project Name Page



CodeWarrior Project Importer - Project Name

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

New Project Name: hcs08_projectimporter_test

☐ Use default location

Location: D:\Project_Importer_Test_projects\hcs08_projectimporter_tes

Choose file system: default

22. To specify a name and location to the imported project, perform these steps.

a. Enter a name for the converted Eclipse project, in the **New Project Name** text box. By default, the old project name is specified.

b. Check **Use default location** to save the project to the default Eclipse workspace. By default, the location of the project is the directory of the classic project and not the default Eclipse workspace.

TIP If you want to save the converted project to a location other than the default Eclipse workspace, click **Browse** and use the **Browse To Folder** dialog box to navigate to the desired directory.

23. Click **Finish**.

    The **CodeWarrior Classic Project Importer** wizard translates the classic CodeWarrior project and the new Eclipse project appears in the **CodeWarrior Project** view of the **Workbench** window.

NOTE Before debugging the new Eclipse project you might need to edit the build and launch configuration settings. For information on build properties, refer to the chapter Build Properties for Bareboard Projects.

Tutorial B: Porting Classic RS08 Project

The goal of this tutorial is to import a classic RS08 project to Eclipse.

NOTE Before starting the process ensure that the CodeWarrior RS08 project you want to import has all of its files, such as the source, linker command, and settings file.

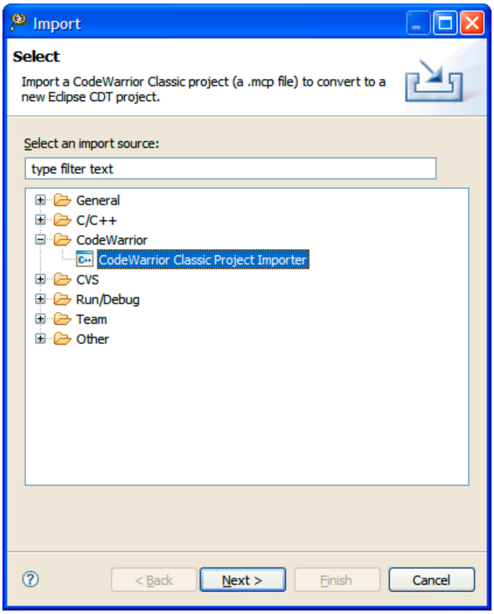
To port a classic RS08 project, perform these steps.

1. Select **File > Import** from the **Workbench** menu bar.

   The **Import** dialog box appears.

2. Expand the **CodeWarrior** tree control and select **CodeWarrior Classic Project Importer**.

Figure 2.105 Import Dialog Box



Import
Select
Select an import source:
type filter text
General
C/C++
CodeWarrior
CodeWarrior Classic Project Importer
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

Import a CodeWarrior Classic project (a .mcp file) to convert to a new Eclipse CDT project.

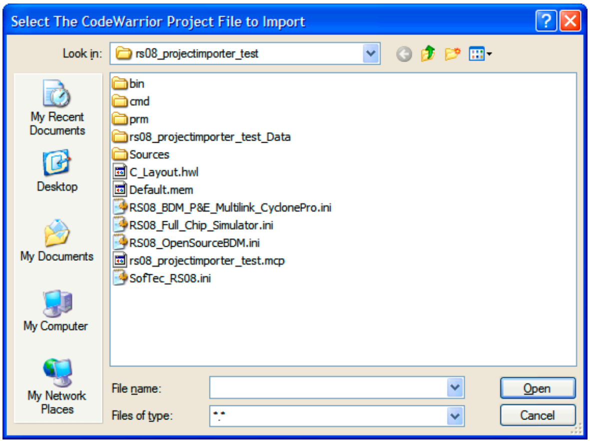
3. Click **Next**.

   The first page of the **CodeWarrior Classic Project Importer** wizard appears.

4. Enter the path and name of the classic CodeWarrior project file to import in the **Project file** text box. Alternatively, click **Browse** and use the **Select The CodeWarrior Project File to Import** dialog box to select the project file to import. In this case, assume that the classic CodeWarrior project filename is `rs08_projectimporter_test.mcp`.

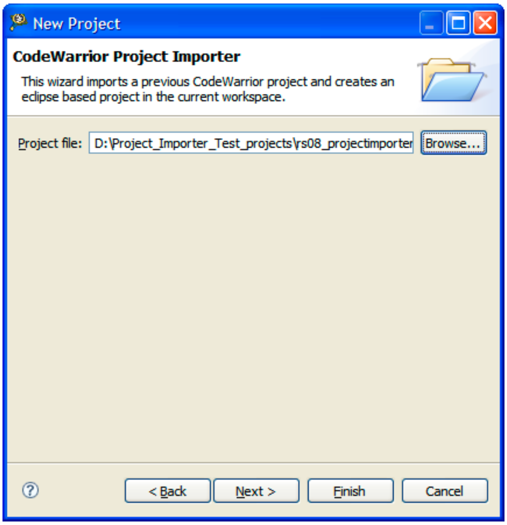
TIP The project file has an extension of `.mcp`. Select the `.mcp` file.

Figure 2.106 Select The CodeWarrior Project File to Import Dialog Box


Select The CodeWarrior Project File to Import
Look in:
rs08_projectimporter_test
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
bin
cmd
prm
rs08_projectimporter_test_Data
Sources
C_Layout.hwl
Default.mem
RS08_BDM_P&E_Multilink_CyclonePro.ini
RS08_Full_Chip_Simulator.ini
RS08_OpenSourceBDM.ini
rs08_projectimporter_test.mcp
SofTec_RS08.ini
File name:
Files of type:
.
Open
Cancel

The path of the project file to import appears in the **Project file** text box.

Figure 2.107 Path of CodeWarrior Project File to Import

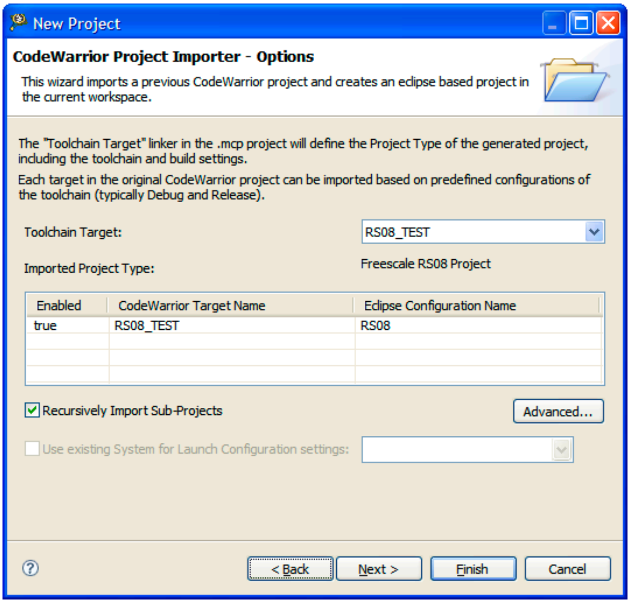

New Project
CodeWarrior Project Importer
This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.
Browse...
< Back
Next >
Finish
Cancel

Project file: D:\Project_Importer_Test_projects\rs08_projectimporter

5. Click **Next**.

   The **Options** page of the **CodeWarrior Classic Project Importer** wizard appears.

Figure 2.108 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: RS08_TEST

Imported Project Type: Freescale RS08 Project

Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	RS08_TEST	RS08

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

6. Select the build target that uses the RS08 toolchain you want the generated Eclipse project to use, from the **Toolchain Target** list box.

NOTE The toolchain target linker in the classic project defines the project type of the generated Eclipse project, including the toolchain and build settings.

The build targets table displays all the targets discovered in the project file and is used to generate equivalent Eclipse build configurations.

7. You can import each build target in the classic CodeWarrior project based upon predefined configurations of the toolchain. For example: The RS08 project in the example lists these values:

- **Imported Project Type** = *Freescale RS08 Project*
- **Enabled** = *true*
- **CodeWarrior Target Name** = <*Toolchain Target*>
- **Eclipse Configuration Name** = *RS08*

TIP To disable the generation of specific configurations, click a row in the build target table. In the **Edit Table Values** dialog box set **Enabled** to *false* and click **OK**.

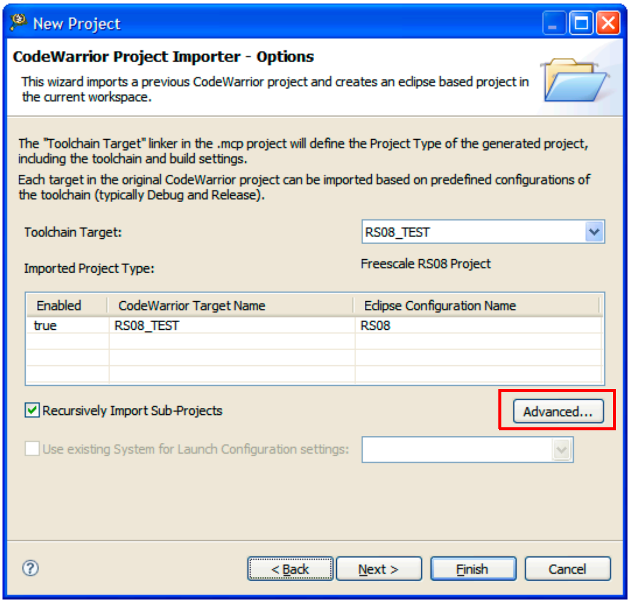
8. If you want to import sub-projects included in the classic CodeWarrior project, check the **Recursively Import Sub-Projects** checkbox.

   The **CodeWarrior Classic Project Importer** wizard imports the sub-projects with the main project.

NOTE The **CodeWarrior Classic Project Importer** wizard will copy only those files that are displayed in the **CodeWarrior Projects** view. The wizard will not import a file if it is not displayed or does not include project information.

9. Click **Advanced**.

Figure 2.109 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: RS08_TEST

Imported Project Type: Freescale RS08 Project

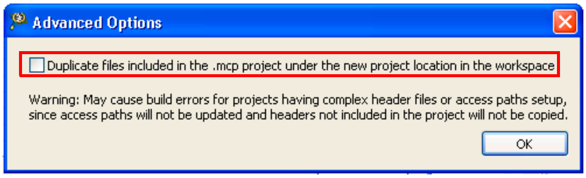
Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	RS08_TEST	RS08

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

The **Advanced Options** dialog box appears.

Figure 2.110 Advanced Options



☐ Duplicate files included in the .mcp project under the new project location in the workspace

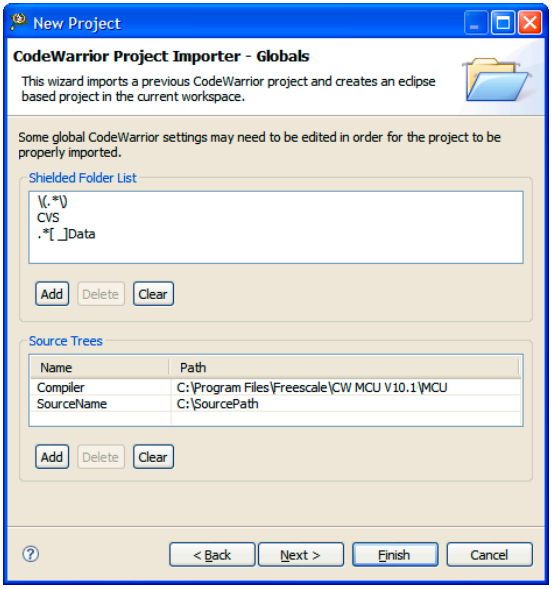
Warning: May cause build errors for projects having complex header files or access paths setup, since access paths will not be updated and headers not included in the project will not be copied.

WARNING! Checking the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox may cause build errors.

10. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox.
11. Click **OK** to close the dialog box.
12. Click **Next**.

    The **Globals** page of the **CodeWarrior Classic Project Importer** wizard appears. This page lets you edit global settings that can effect how the project's build options are imported.

Figure 2.111 CodeWarrior Project Importer — Globals Page



CodeWarrior Project Importer - Globals

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Some global CodeWarrior settings may need to be edited in order for the project to be properly imported.

Shielded Folder List

```
\(.*\)
CVS
.*[ _]Data
```

Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

Table 2.23 lists the options on the **CodeWarrior Classic Project Importer — Globals** page.

Table 2.23 CodeWarrior Project Importer — Globals Page Options

Options	Description
Shielded Folder List	Lists the concealed contents of folders from the IDE's search operations during a build. This was done by placing special characters in the directory name. For example, sample code was concealed in a (CodeWarrior Examples) folder. The **Shielded Folder List** table lists these options: • \(.*\) • CVS • .*[_]Data You use the **Add, Delete**, and **Clear** buttons to modify the information in this list. Table 2.24 lists these buttons with their descriptions.
Sources	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The **{Project}** source tree is defined automatically. The **Sources Trees** table lists these options: • Name — Lists the source name. For example: Compiler. • Compiler — Lists the path source name. For example: `<CW Install>/MCU`. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.24 lists these buttons with their descriptions.

Table 2.24 CodeWarrior Project Importer — Globals Page Buttons

Button	Description
Add	Add a new entry to the list.
Delete	Deletes the selected item.
Clear	Clears the entire list.

13. To add a new expression to the **Shielded Folder List** table, perform these steps.

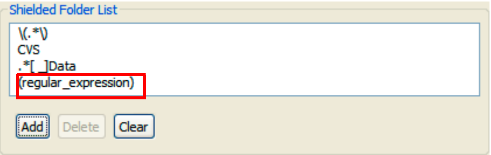
a. Click **Add**.

(regular_expression) appears in the shielded folder list.

b. Double-click *(regular_expression)* and type the required expression.

The new expression appears in shielded folder list.

Figure 2.112 Shielded Folder List



```
\(.*\)
CVS
.*[ _]Data
(regular_expression)
```

14. To delete an existing expression from the **Shielded Folder List** table, select the expression and click **Delete**.

    The selected expression is deleted from the shielded folder list.

15. To remove all the existing expressions from the **Shielded Folder List** table, click **Clear**.

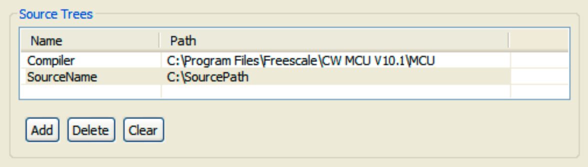
    All the expressions are deleted from the shielded folder list.

16. To add a new source to the **Source Trees** table, perform these steps.

    a. Click **Add**.

    SourceName appears in the source trees list.

Figure 2.113 Sources Trees

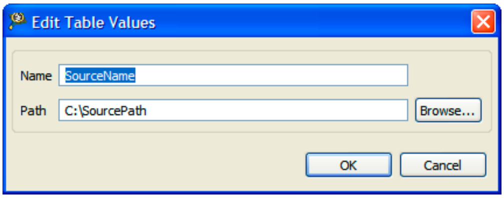


Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

b. Double-click *SourceName.*

The **Edit Table Values** dialog box appears.

Figure 2.114 Edit Table Values



Name: SourceName
Path: C:\SourcePath

c. In the **Name** text box, enter the source name.

d. In the **Path** text box, enter the path of the new source. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

e. Click **OK**.

The new source appears in **Source Trees** list.

17. Click **Next**.

The **CodeWarrior Project Importer - Access Paths** page appears.

NOTE Access paths are directory paths the CodeWarrior tools use to search for libraries, runtime support files, and other object files.

Table 2.25 lists the options on the **CodeWarrior Project Importer — Access Paths** page.

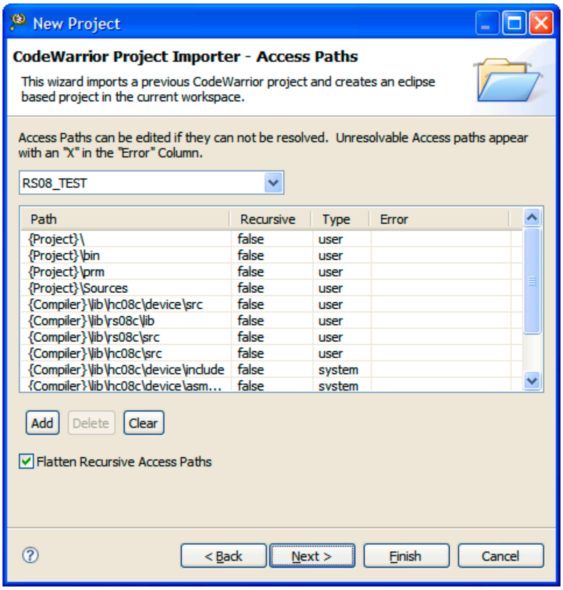
Table 2.25 CodeWarrior Project Importer — Access Paths Page

Option	Description
Build Target List Box	Lets you select the build target whose access paths you want to modify. For example: `RS08_TEST`

Table 2.25 CodeWarrior Project Importer — Access Paths Page (*continued*)

Option	Description
Access Path Table	Lists the access paths used by the build target selected in the **Build Target** list box. Each row lists: • **Path** — Directory path. For example: The path is `{Compiler}\lib\rs08c\include.` • **Recursive** — Whether the path is searched recursively. For example: `false` or `true.` • **Type** — Specify of path to be searched. For example: `user` or `system.` • **Error** — Unresolved access paths, marked as “X”, if any. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.26 lists these buttons with their descriptions.
Flatten Recursive Access Paths Checkbox	When checked, the **CodeWarrior Classic Project Importer** wizard automatically generates separate include paths for each subdirectory that is a part of the recursive path. This option is set by default as most compilers do not support recursive include paths passed by the command line.

Figure 2.115 CodeWarrior Project Importer — Access Paths Page



This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Access Paths can be edited if they can not be resolved. Unresolvable Access paths appear with an "X" in the "Error" Column.

RS08_TEST

Path	Recursive	Type	Error
{Project}\	false	user	
{Project}\bin	false	user	
{Project}\prm	false	user	
{Project}\Sources	false	user	
{Compiler}\lib\hc08c\device\src	false	user	
{Compiler}\lib\rs08c\lib	false	user	
{Compiler}\lib\rs08c\src	false	user	
{Compiler}\lib\hc08c\src	false	user	
{Compiler}\lib\hc08c\device\include	false	system	
{Compiler}\lib\hc08c\device\asm...	false	system	

☑ Flatten Recursive Access Paths

Table 2.26 CodeWarrior Project Importer — Access Paths Page Buttons

Button	Description
Add	Add a new directory path to the list.
Delete	Deletes the selected directory path from the list.
Clear	Clears the entire list.

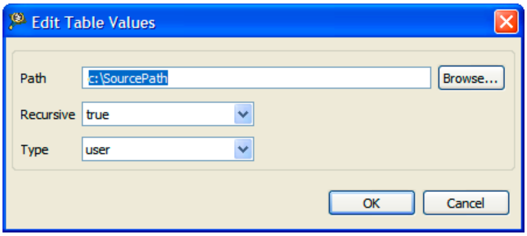
18. To add a new directory path to list, perform these steps.
    a. Click **Add**.

       (C:\SourcePath) appears in the **Path** list.

    b. Click *(C:\SourcePath)*.

       The **Edit Table Values** dialog box appears.

Figure 2.116 Edit Table Values



Path: c:\SourcePath
Recursive: true
Type: user

c. In the **Path** text box, enter the directory path. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

d. From the **Recursive** list box, select `false` or `true`.

e. From the **Type** list box, select `user` or `system`.

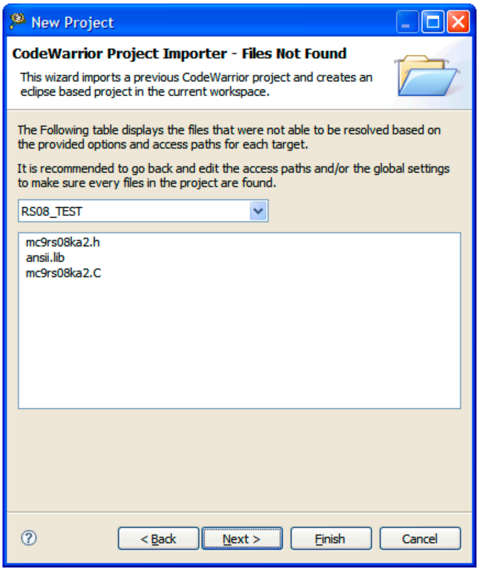
f. Click **OK**.

The new access path appears in table.

19. Click **Next**.

The **CodeWarrior Project Importer — Files Not Found** page appears. This page displays the project files that the wizard could not locate. You can use the Build Target list box to select another build target and view the missing files.

Figure 2.117 CodeWarrior Project Importer — Files Not Found Page



CodeWarrior Project Importer - Files Not Found

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

RS08_TEST

mc9rs08ka2.h
ansii.lib
mc9rs08ka2.C

20. To locate the missing files, perform these steps.

a. Click **Back** to adjust the settings in the **Globals and Access Paths** pages so that the wizard can locate the missing files.

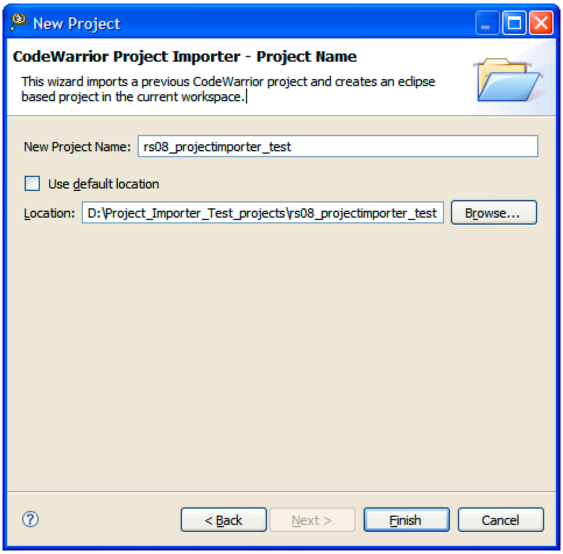
b. Repeat till you narrow down the number of missing files.

NOTE Some old files do not work with the <*target*> implementation, there will be some files missing.

21. Click **Next**.

The **CodeWarrior Project Importer - Project Name** page appears. This page lets you specify the name and select a location for the newly imported project.

Figure 2.118 CodeWarrior Project Importer — Project Name Page



CodeWarrior Project Importer - Project Name

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

New Project Name: rs08_projectimporter_test

☐ Use default location

Location: D:\Project_Importer_Test_projects\rs08_projectimporter_test

22. To specify a name and location to the imported project, perform these steps.

a. Enter a name for the converted Eclipse project, in the **New Project Name** text box. By default, the old project name is specified.

b. Check **Use default location** to save the project to the default Eclipse workspace. By default, the location of the project is the directory of the classic project and not the default Eclipse workspace.

TIP If you want to save the converted project to a location other than the default Eclipse workspace, click **Browse** and use the **Browse To Folder** dialog box to navigate to the desired directory.

23. Click **Finish**.

    The **CodeWarrior Classic Project Importer** wizard translates the classic CodeWarrior project and the new Eclipse project appears in the **CodeWarrior Project** view of the **Workbench** window.

NOTE Before debugging the new Eclipse project you might need to edit the build and launch configuration settings. For information on build properties, refer to the chapter Build Properties for Bareboard Projects.

Tutorial C: Porting Classic ColdFire V1 Project

The goal of this tutorial is to import a classic ColdFire V1 project to Eclipse.

NOTE Before starting the process ensure that the ColdFire V1 project you want to import has all of its files, such as the source, linker command, and settings file.

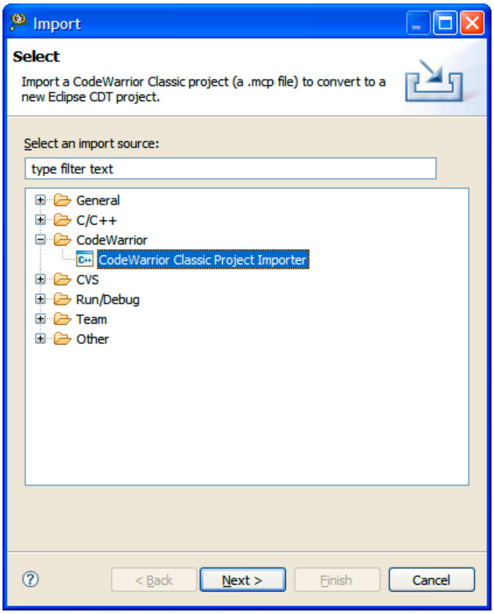
To port a classic ColdFire V1 project, perform these steps.

1. Select **File > Import** from the **Workbench** menu bar.

   The **Import** dialog box appears.

2. Expand the **CodeWarrior** tree control and select **CodeWarrior Classic Project Importer**.

Figure 2.119 Import Dialog Box


Import
Select
Import a CodeWarrior Classic project (a .mcp file) to convert to a new Eclipse CDT project.
Select an import source:
type filter text
General
C/C++
CodeWarrior
CodeWarrior Classic Project Importer
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

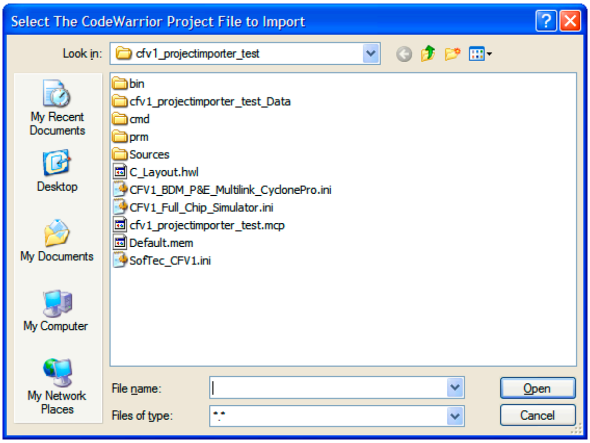
3. Click **Next**.

   The first page of the **CodeWarrior Classic Project Importer** wizard appears.

4. Enter the path and name of the classic CodeWarrior project file to import in the **Project file** text box. Alternatively, click **Browse** and use the **Select The CodeWarrior Project File to Import** dialog box to select the project file to import. In this case, assume that the classic CodeWarrior project filename is `cfv1_projectimporter_test.mcp`.

TIP The project file has an extension of `.mcp`. Select the `.mcp` file.

Figure 2.120 Select The CodeWarrior Project File to Import Dialog Box

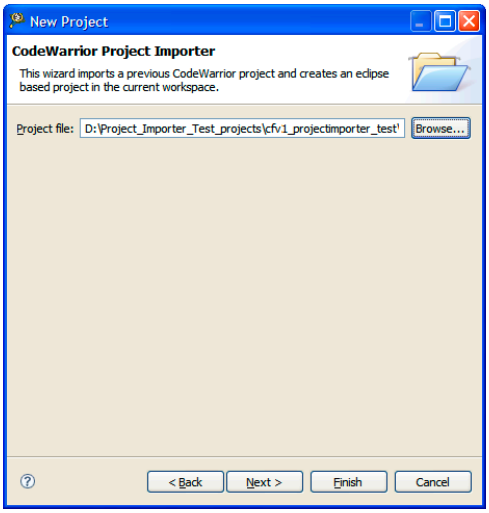

Select The CodeWarrior Project File to Import
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
bin
cfv1_projectimporter_test_Data
cmd
prm
Sources
C_Layout.hwl
CFV1_BDM_P&E_Multilink_CyclonePro.ini
CFV1_Full_Chip_Simulator.ini
cfv1_projectimporter_test.mcp
Default.mem
SofTec_CFV1.ini
File name:
Open
Cancel

Look in: cfv1_projectimporter_test

Files of type: *.*

The path of the project file to import appears in the **Project file** text box.

Figure 2.121 Path of CodeWarrior Project File to Import


New Project
Browse...
< Back
Next >
Finish
Cancel

CodeWarrior Project Importer

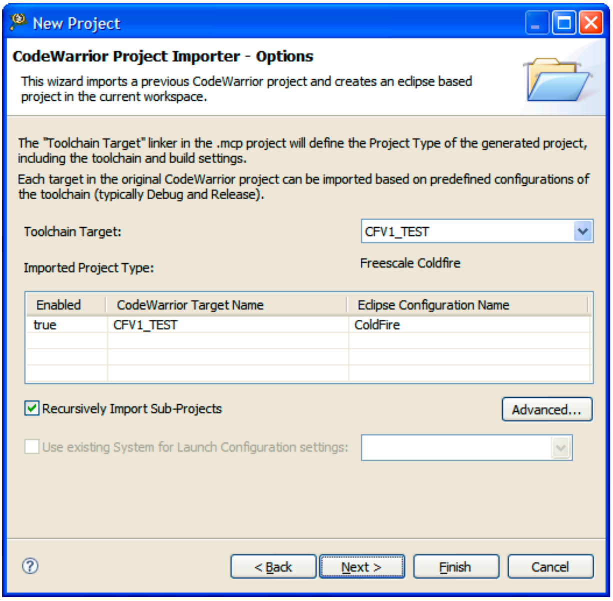
This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Project file: D:\Project_Importer_Test_projects\cfv1_projectimporter_test\

5. Click **Next**.

   The **Options** page of the **CodeWarrior Classic Project Importer** wizard appears.

Figure 2.122 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: CFV1_TEST

Imported Project Type: Freescale Coldfire

Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	CFV1_TEST	ColdFire

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

6. Select the build target that uses the ColdFire toolchain you want the generated Eclipse project to use, from the **Toolchain Target** list box.

NOTE The toolchain target linker in the classic project defines the project type of the generated Eclipse project, including the toolchain and build settings.

The build targets table displays all the targets discovered in the project file and is used to generate equivalent Eclipse build configurations.

7. You can import each build target in the classic CodeWarrior project based upon predefined configurations of the toolchain. For example: The ColdFire project in the example lists these values:
   - **Imported Project Type** = *Freescale ColdFire*
   - **Enabled** = *true*
   - **CodeWarrior Target Name** = <*Toolchain Target*>
   - **Eclipse Configuration Name** = *ColdFire*

TIP To disable the generation of specific configurations, click a row in the build target table. In the **Edit Table Values** dialog box set **Enabled** to *false* and click **OK**.

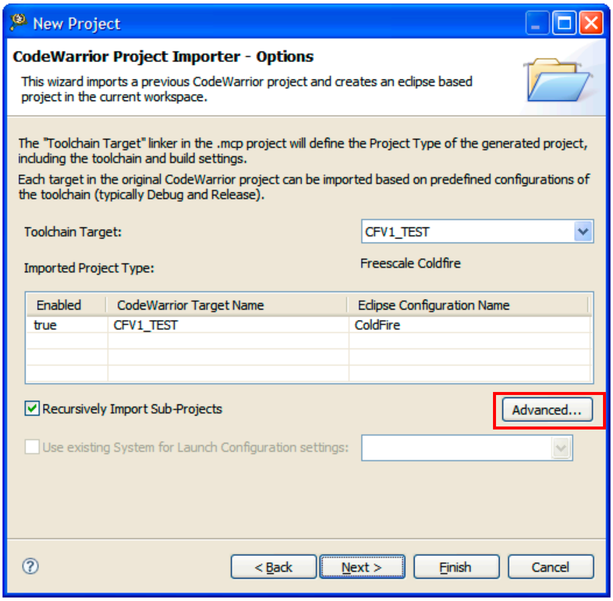
8. If you want to import sub-projects included in the classic CodeWarrior project, check the **Recursively Import Sub-Projects** checkbox.

   The **CodeWarrior Project Importer** wizard imports the sub-projects with the main project.

NOTE The **CodeWarrior Classic Project Importer** wizard will copy only those files that are displayed in the **CodeWarrior Projects** view. The wizard will not import a file if it is not displayed or does not include project information.

9. Click **Advanced**.

Figure 2.123 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: CFV1_TEST

Imported Project Type: Freescale Coldfire

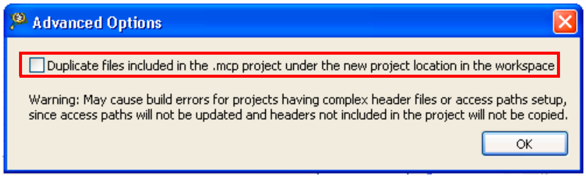
Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	CFV1_TEST	ColdFire

☑ Recursively Import Sub-Projects Advanced...

☐ Use existing System for Launch Configuration settings:

The **Advanced Options** dialog box appears.

Figure 2.124 Advanced Options



☐ Duplicate files included in the .mcp project under the new project location in the workspace

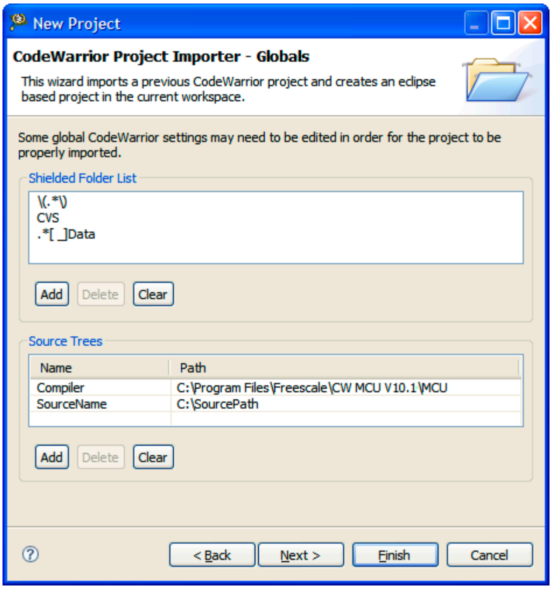
Warning: May cause build errors for projects having complex header files or access paths setup, since access paths will not be updated and headers not included in the project will not be copied.

WARNING! Checking the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox may cause build errors.

10. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox.
11. Click **OK** to close the dialog box.
12. Click **Next**.

    The **Globals** page of the **CodeWarrior Project Importer** wizard appears. This page lets you edit global settings that can effect how the project's build options are imported.

Figure 2.125 CodeWarrior Classic Project Importer — Globals Page



CodeWarrior Project Importer - Globals

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Some global CodeWarrior settings may need to be edited in order for the project to be properly imported.

Shielded Folder List

```
\(.*\)
CVS
.*[ _]Data
```

Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

Table 2.27 lists the options on the **CodeWarrior Project Importer — Globals** page.

Table 2.27 CodeWarrior Project Importer — Globals Page Options

Options	Description
Shielded Folder List	Previous versions of the CodeWarrior tools let you conceal the contents of folders from the IDE's search operations during a build. This was done by placing special characters in the directory name. For example, sample code was concealed in a (CodeWarrior Examples) folder. The **Shielded Folder List** table lists these options: • \(.*\) • CVS • .*[_]Data You use the **Add, Delete**, and **Clear** buttons to modify the information in this list. Table 2.28 lists these buttons with their descriptions.
Sources	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The **{Project}** source tree is defined automatically. The **Sources Trees** table lists these options: • Name — Lists the source name. For example: Compiler. • Compiler — Lists the path source name. For example: `<CW Install>/MCU.` You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.28 lists these buttons with their descriptions.

Table 2.28 CodeWarrior Project Importer — Globals Page Buttons

Button	Description
Add	Add a new entry to the list.
Delete	Deletes the selected item.
Clear	Clears the entire list.

13. To add a new expression to the **Shielded Folder List** table, perform these steps.

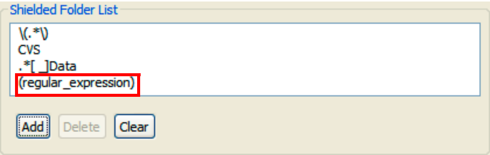
a. Click **Add**.

(regular_expression) appears in the shielded folder list.

b. Double-click *(regular_expression)* and type the required expression.

The new expression appears in shielded folder list.

Figure 2.126 Shielded Folder List



Shielded Folder List

\(.*\)
CVS
.*[_]Data
(regular_expression)

14. To delete an existing expression from the **Shielded Folder List** table, select the expression and click **Delete**.

    The selected expression is deleted from the shielded folder list.

15. To remove all the existing expressions from the **Shielded Folder List** table, click **Clear**.

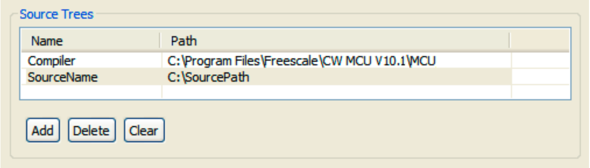
    All the expressions are deleted from the shielded folder list.

16. To add a new source to the **Source Trees** table, perform these steps.

    a. Click **Add**.

    SourceName appears in the source trees list.

Figure 2.127 Sources Trees



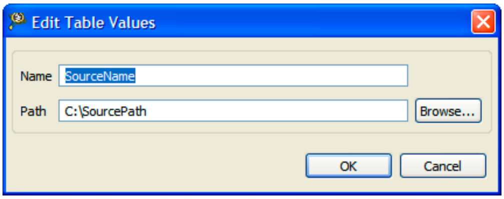
Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

b. Double-click *SourceName*.

The **Edit Table Values** dialog box appears.

Figure 2.128 Edit Table Values



Name: SourceName
Path: C:\SourcePath

c. In the **Name** text box, enter the source name.

d. In the **Path** text box, enter the path of the new source. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

e. Click **OK**.

The new source appears in **Source Trees** list.

17. Click **Next**.

The **CodeWarrior Project Importer - Access Paths** page appears.

NOTE Access paths are directory paths the CodeWarrior tools use to search for libraries, runtime support files, and other object files.

Table 2.29 lists the options on the **CodeWarrior Project Importer — Access Paths** page.

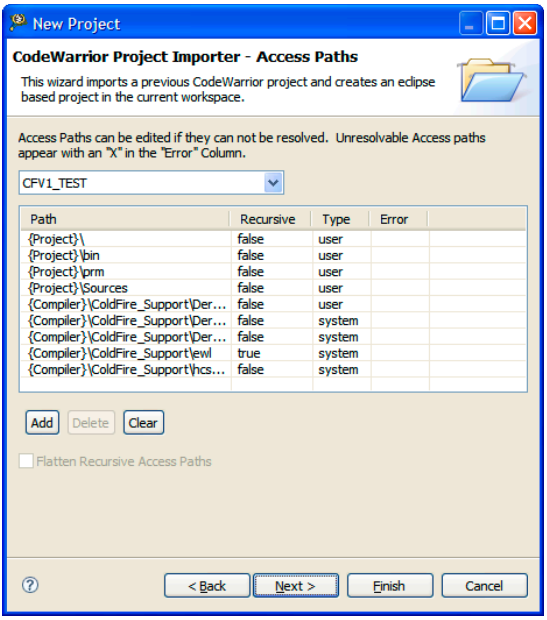
Table 2.29 CodeWarrior Project Importer — Access Paths Page

Option	Description
Build Target List Box	Lets you select the build target whose access paths you want to modify. For example: `CFV1_TEST`

Table 2.29 CodeWarrior Project Importer — Access Paths Page (*continued*)

Option	Description
Access Path Table	Lists the access paths used by the build target selected in the **Build Target** list box. Each row lists: • **Path** — Directory path. For example: The path is `{Compiler}\ColdFire_Support\hcs08_compatibility`. • **Recursive** — Whether the path is searched recursively. For example: `false` or `true`. • **Type** — Specify of path to be searched. For example: `user` or `system`. • **Error** — Unresolved access paths, marked as “X”, if any. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.30 lists these buttons with their descriptions.
Flatten Recursive Access Paths Checkbox	When checked, the **CodeWarrior Classic Project Importer** wizard automatically generates separate include paths for each subdirectory that is a part of the recursive path. This option is set by default as most compilers do not support recursive include paths passed by the command line.

Figure 2.129 CodeWarrior Project Importer — Access Paths Page



CodeWarrior Project Importer - Access Paths

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Access Paths can be edited if they can not be resolved. Unresolvable Access paths appear with an "X" in the "Error" Column.

CFV1_TEST

Path	Recursive	Type	Error
{Project}\	false	user	
{Project}\bin	false	user	
{Project}\prm	false	user	
{Project}\Sources	false	user	
{Compiler}\ColdFire_Support\Der...	false	user	
{Compiler}\ColdFire_Support\Der...	false	system	
{Compiler}\ColdFire_Support\Der...	false	system	
{Compiler}\ColdFire_Support\ewl	true	system	
{Compiler}\ColdFire_Support\hcs...	false	system	

Table 2.30 CodeWarrior Project Importer — Access Paths Page Buttons

Button	Description
Add	Add a new directory path to the list.
Delete	Deletes the selected directory path from the list.
Clear	Clears the entire list.

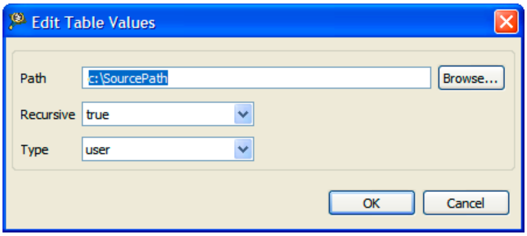
18. To add a new directory path to list, perform these steps.
    a. Click **Add**.

       (C:\SourcePath) appears in the **Path** list.

    b. Click *(C:\SourcePath)*.

       The **Edit Table Values** dialog box appears.

Figure 2.130 Edit Table Values



Edit Table Values

Path: c:\SourcePath

Recursive: true

Type: user

c. In the **Path** text box, enter the directory path. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

d. From the **Recursive** list box, select `false` or `true`.

e. From the **Type** list box, select `user` or `system`.

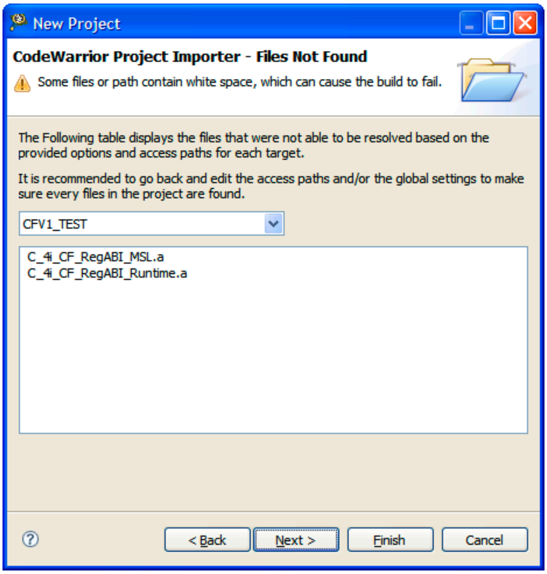
f. Click **OK**.

The new access path appears in table.

19. Click **Next**.

The **CodeWarrior Project Importer — Files Not Found** page appears. This page displays the project files that the wizard could not locate. You can use the Build Target list box to select another build target and view the missing files.

Figure 2.131 CodeWarrior Project Importer — Files Not Found Page



New Project

CodeWarrior Project Importer - Files Not Found

Some files or path contain white space, which can cause the build to fail.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

CFV1_TEST

C_4i_CF_RegABI_MSL.a
C_4i_CF_RegABI_Runtime.a

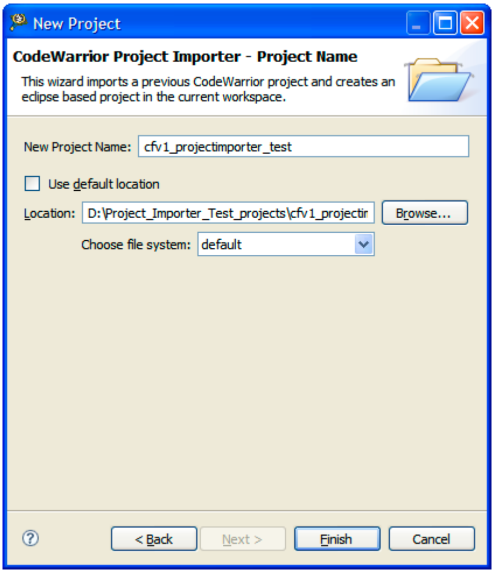
20. To locate the missing files, peform these steps.
    a. Click **Back** to adjust the settings in the **Globals and Access Paths** pages so that the wizard can locate the missing files.
    b. Repeat till you narrow down the number of missing files.

NOTE Some old files do not work with the <*target*> implementation, there will be some files missing.

21. Click **Next**.

    The **CodeWarrior Project Importer - Project Name** page appears. This page lets you specify the name and select a location for the newly imported project.

Figure 2.132 CodeWarrior Classic Project Importer — Project Name Page



CodeWarrior Project Importer - Project Name

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

New Project Name: cfv1_projectimporter_test

☐ Use default location

Location: D:\Project_Importer_Test_projects\cfv1_projectir

Choose file system: default

22. To specify a name and location to the imported project, perform these steps.
    a. Enter a name for the converted Eclipse project, in the **New Project Name** text box. By default, the old project name is specified.
    b. Check **Use default location** to save the project to the default Eclipse workspace. By default, the location of the project is the directory of the classic project and not the default Eclipse workspace.

TIP If you want to save the converted project to a location other than the default Eclipse workspace, click **Browse** and use the **Browse To Folder** dialog box to navigate to the desired directory.

23. Click **Finish**.

    The **CodeWarrior Classic Project Importer** wizard translates the classic CodeWarrior project and the new Eclipse project appears in the **CodeWarrior Project** view of the **Workbench** window.

NOTE Before debugging the new Eclipse project you might need to edit the build and launch configuration settings. For information on build properties, refer to the chapter Build Properties for Bareboard Projects.

Tutorial D: Porting Classic ColdFire V2/3/4 Project

The goal of this tutorial is to import a classic ColdFireV2/3/4 project to Eclipse.

NOTE Before starting the process ensure that the CodeWarrior V2/3/4 project you want to import has all of its files, such as the source, linker command, and settings file.

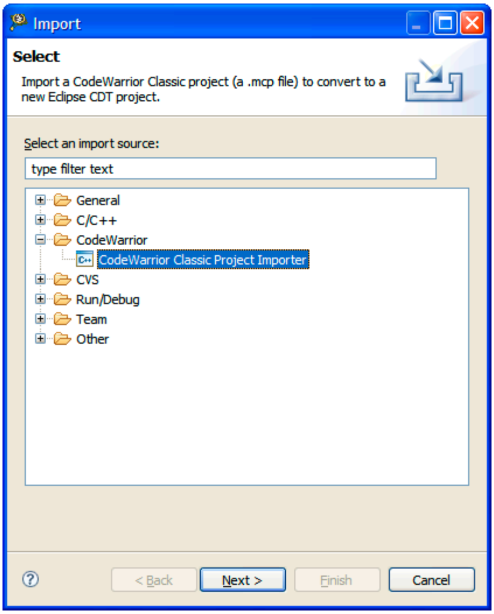
To port a classic ColdFire V2/3/4 project, perform these steps.

1. Select **File > Import** from the **Workbench** menu bar.

   The **Import** dialog box appears.

2. Expand the **CodeWarrior** tree control and select **CodeWarrior Classic Project Importer**.

Figure 2.133 Import Dialog Box



Select

Import a CodeWarrior Classic project (a .mcp file) to convert to a new Eclipse CDT project.

Import
Select an import source:
type filter text
General
C/C++
CodeWarrior
CodeWarrior Classic Project Importer
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

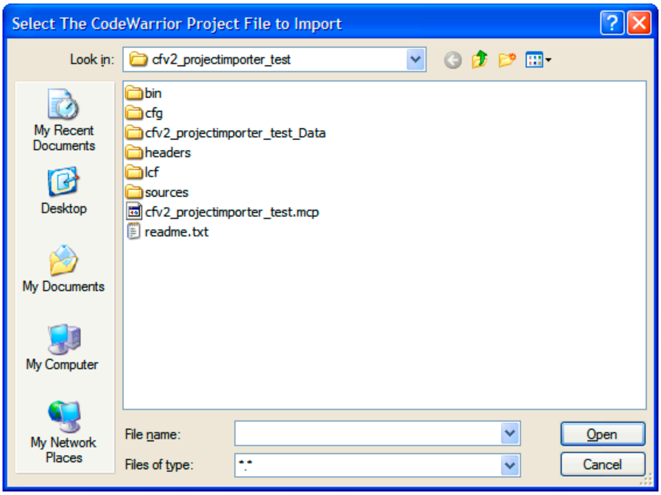
3. Click **Next**.

   The first page of the **CodeWarrior Classic Project Importer** wizard appears.

4. Enter the path and name of the classic CodeWarrior project file to import in the **Project file** text box. Alternatively, click **Browse** and use the **Select The CodeWarrior Project File to Import** dialog box to select the project file to import. In this case, assume that the classic CodeWarrior project filename is `cfv2_projectimporter_test.mcp`.

TIP The project file has an extension of `.mcp`. Select the `.mcp` file.

Figure 2.134 Select The CodeWarrior Project File to Import Dialog Box


Select The CodeWarrior Project File to Import
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
bin
cfg
cfv2_projectimporter_test_Data
headers
lcf
sources
cfv2_projectimporter_test.mcp
readme.txt
Open
Cancel

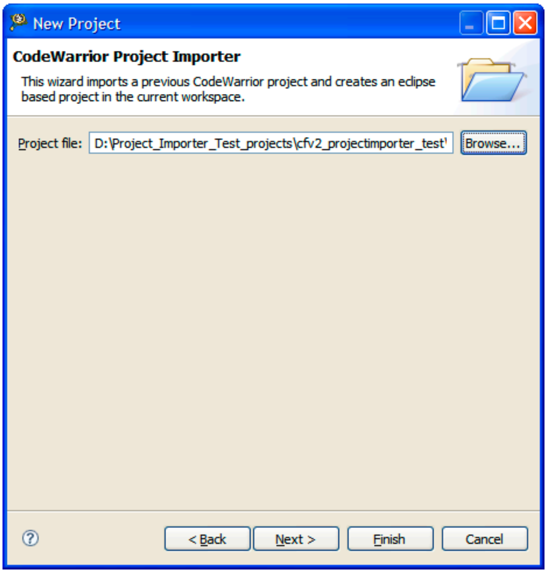
Look in: cfv2_projectimporter_test

File name:

Files of type: *.*

The path of the project file to import appears in the **Project file** text box.

Figure 2.135 Path of CodeWarrior Project File to Import


New Project
CodeWarrior Project Importer
Browse...
< Back
Next >
Finish
Cancel

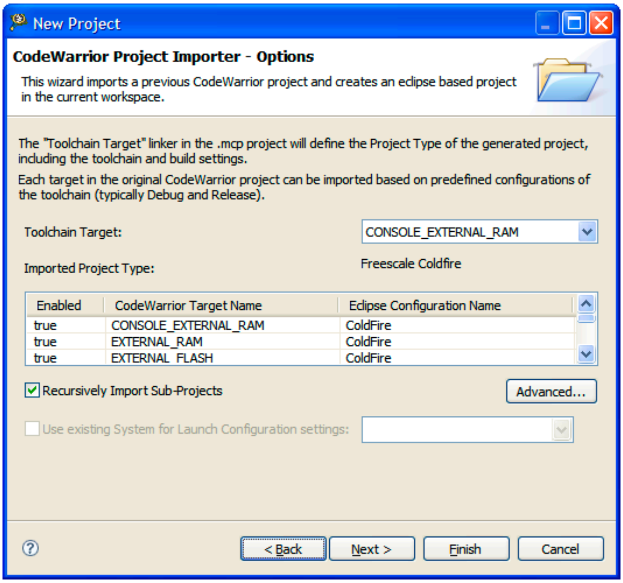
This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Project file: D:\Project_Importer_Test_projects\cfv2_projectimporter_test\

5. Click **Next**.

   The **Options** page of the **CodeWarrior Project Importer** wizard appears.

Figure 2.136 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: CONSOLE_EXTERNAL_RAM

Imported Project Type: Freescale Coldfire

Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	CONSOLE_EXTERNAL_RAM	ColdFire
true	EXTERNAL_RAM	ColdFire
true	EXTERNAL_FLASH	ColdFire

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

6. Select the build target that uses the ColdFire toolchain you want the generated Eclipse project to use, from the **Toolchain Target** list box.

NOTE The toolchain target linker in the classic project defines the project type of the generated Eclipse project, including the toolchain and build settings.

The build targets table displays all the targets discovered in the project file and is used to generate equivalent Eclipse build configurations.

7. You can import each build target in the classic CodeWarrior project based upon predefined configurations of the toolchain. For example: The ColdFire project in the example lists these values:
   - **Imported Project Type** = *Freescale ColdFire*
   - **Enabled** = *true*
   - **CodeWarrior Target Name** = <*Toolchain Targets*>
   - **Eclipse Configuration Name** = *ColdFire*

TIP To disable the generation of specific configurations, click a row in the build target table. In the **Edit Table Values** dialog box set **Enabled** to *false* and click **OK**.

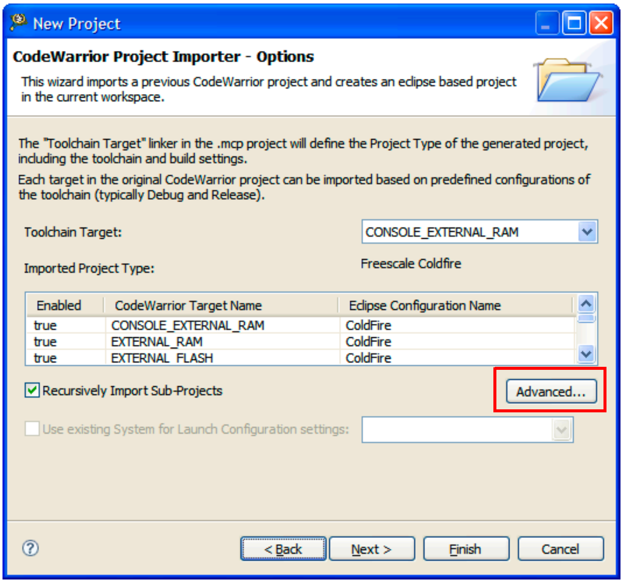
8. If you want to import sub-projects included in the classic CodeWarrior project, check the **Recursively Import Sub-Projects** checkbox.

   The **CodeWarrior Project Importer** wizard imports the sub-projects with the main project.

NOTE The **CodeWarrior Project Importer** wizard will copy only those files that are displayed in the **CodeWarrior Projects** view. The wizard will not import a file if it is not displayed or does not include project information.

9. Click **Advanced**.

Figure 2.137 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: CONSOLE_EXTERNAL_RAM

Imported Project Type: Freescale Coldfire

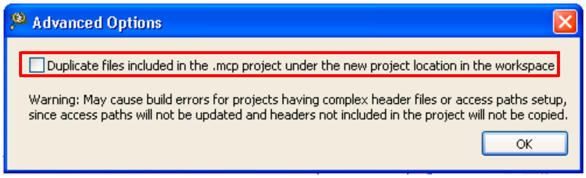
Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	CONSOLE_EXTERNAL_RAM	ColdFire
true	EXTERNAL_RAM	ColdFire
true	EXTERNAL_FLASH	ColdFire

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

The **Advanced Options** dialog box appears.

Figure 2.138 Advanced Options



☐ Duplicate files included in the .mcp project under the new project location in the workspace

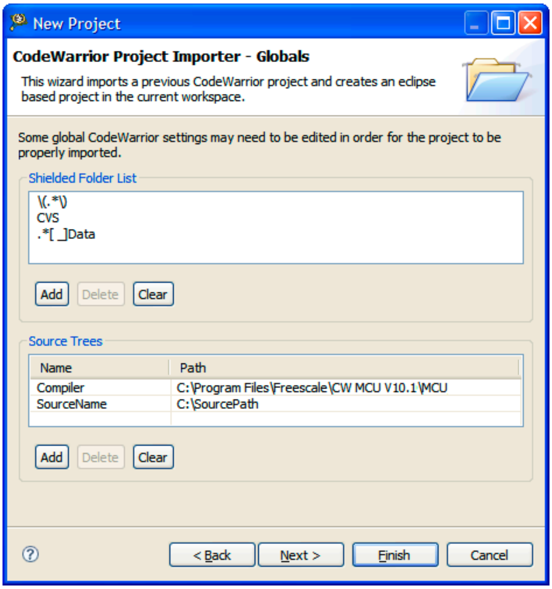
Warning: May cause build errors for projects having complex header files or access paths setup, since access paths will not be updated and headers not included in the project will not be copied.

WARNING! Checking the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox may cause build errors.

10. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox.
11. Click **OK** to close the dialog box.
12. Click **Next**.

    The **Globals** page of the **CodeWarrior Project Importer** wizard appears. This page lets you edit global settings that can effect how the project's build options are imported.

Figure 2.139 CodeWarrior Project Importer — Globals Page



CodeWarrior Project Importer - Globals

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Some global CodeWarrior settings may need to be edited in order for the project to be properly imported.

Shielded Folder List

```
\(.*\)
CVS
.*[ _]Data
```

Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

[Table 2.31](#) lists the options on the **CodeWarrior Project Importer — Globals** page.

Table 2.31 CodeWarrior Project Importer — Globals Page Options

Options	Description
Shielded Folder List	Previous versions of the CodeWarrior tools let you conceal the contents of folders from the IDE's search operations during a build. This was done by placing special characters in the directory name. For example, sample code was concealed in a (CodeWarrior Examples) folder. The **Shielded Folder List** table lists these options: • \(.*\) • CVS • .*[_]Data You use the **Add, Delete**, and **Clear** buttons to modify the information in this list. Table 2.32 lists these buttons with their descriptions.
Sources	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The **{Project}** source tree is defined automatically. The **Sources Trees** table lists these options: • Name — Lists the source name. For example: Compiler. • Compiler — Lists the path source name. For example: `<CW Install>/MCU`. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.32 lists these buttons with their descriptions.

Table 2.32 CodeWarrior Project Importer — Globals Page Buttons

Button	Description
Add	Add a new entry to the list.
Delete	Deletes the selected item.
Clear	Clears the entire list.

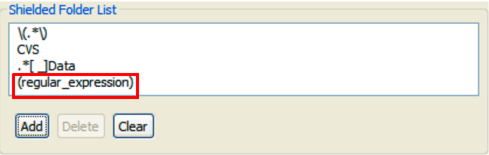
a. Click **Add**.

(regular_expression) appears in the shielded folder list.

b. Double-click *(regular_expression)* and type the required expression.

The new expression appears in shielded folder list.

Figure 2.140 Shielded Folder List



\(.*\)
CVS
.*[_]Data
(regular_expression)

13. To delete an existing expression from the **Shielded Folder List** table, select the expression and click **Delete**.

    The selected expression is deleted from the shielded folder list.

14. To remove all the existing expressions from the **Shielded Folder List** table, click **Clear**.

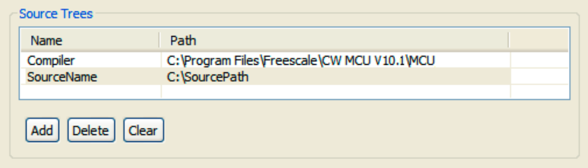
    All the expressions are deleted from the shielded folder list.

15. To add a new source to the **Source Trees** table, perform these steps.

    a. Click **Add**.

    SourceName appears in the source trees list.

Figure 2.141 Sources Trees

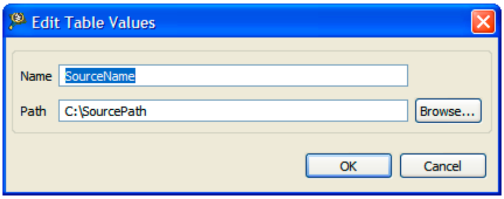


Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

b. Double-click *SourceName.*

The **Edit Table Values** dialog box appears.

Figure 2.142 Edit Table Values



Name: SourceName
Path: C:\SourcePath

c. In the **Name** text box, enter the source name.

d. In the **Path** text box, enter the path of the new source. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

e. Click **OK**.

The new source appears in **Source Trees** list.

16. Click **Next**.

The **CodeWarrior Project Importer - Access Paths** page appears.

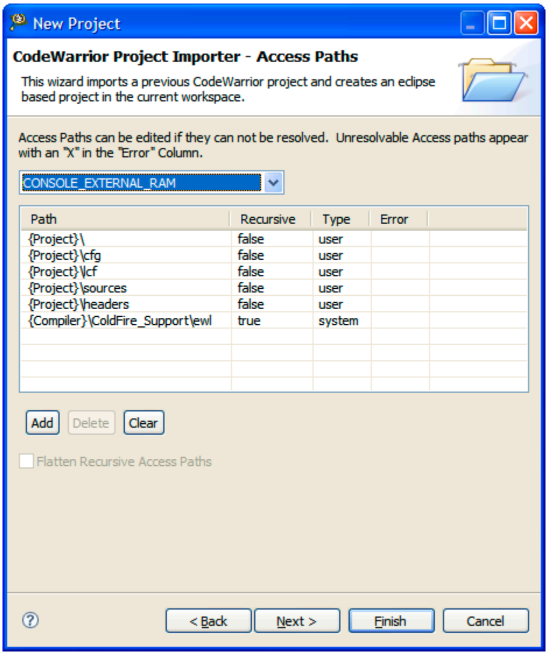
NOTE Access paths are directory paths the CodeWarrior tools use to search for libraries, runtime support files, and other object files.

Table 2.33 lists the options on the **CodeWarrior Project Importer — Access Paths** page.

Table 2.33 CodeWarrior Project Importer — Access Paths Page

Option	Description
Build Target List Box	Lets you select the build target whose access paths you want to modify. For example: `CONSOLE_EXTERNAL_RAM`.
Access Path Table	Lists the access paths used by the build target selected in the **Build Target** list box. Each row lists: • **Path** — Directory path. For example: The path is `{Compiler}\ColdFire_Support\ewl`. • **Recursive** — Whether the path is searched recursively. For example: `false` or `true`. • **Type** — Specify of path to be searched. For example: `user` or `system`. • **Error** — Unresolved access paths, marked as “X”, if any. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.34 lists these buttons with their descriptions.
Flatten Recursive Access Paths Checkbox	When checked, the **CodeWarrior Project Importer** wizard automatically generates separate include paths for each subdirectory that is a part of the recursive path. This option is set by default as most compilers do not support recursive include paths passed by the command line.

Figure 2.143 CodeWarrior Project Importer — Access Paths Page



CodeWarrior Project Importer - Access Paths

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Access Paths can be edited if they can not be resolved. Unresolvable Access paths appear with an "X" in the "Error" Column.

CONSOLE_EXTERNAL_RAM

Path	Recursive	Type	Error
{Project}\	false	user	
{Project}\cfg	false	user	
{Project}\lcf	false	user	
{Project}\sources	false	user	
{Project}\headers	false	user	
{Compiler}\ColdFire_Support\ewl	true	system	

☐ Flatten Recursive Access Paths

Table 2.34 CodeWarrior Project Importer — Access Paths Page Buttons

Button	Description
Add	Add a new directory path to the list.
Delete	Deletes the selected directory path from the list.
Clear	Clears the entire list.

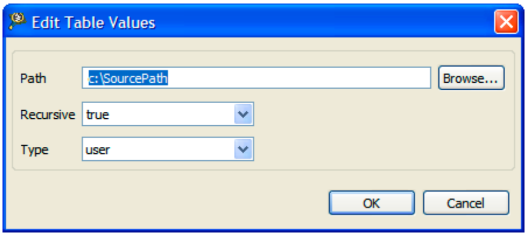
17. To add a new directory path to list, perform these steps.
    a. Click **Add**.

       (C:\SourcePath) appears in the **Path** list.

    b. Click *(C:\SourcePath)*.

       The **Edit Table Values** dialog box appears.

Figure 2.144 Edit Table Values



Path: c:\SourcePath
Recursive: true
Type: user

c. In the **Path** text box, enter the directory path. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

d. From the **Recursive** list box, select `false` or `true`.

e. From the **Type** list box, select `user` or `system`.

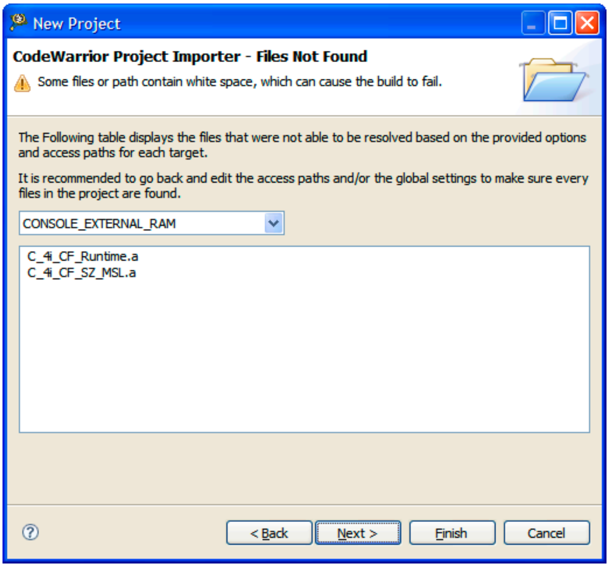
f. Click **OK**.

The new access path appears in table.

18. Click **Next**.

The **CodeWarrior Project Importer — Files Not Found** page appears. This page displays the project files that the wizard could not locate. You can use the Build Target list box to select another build target and view the missing files.

Figure 2.145 CodeWarrior Project Importer — Files Not Found Page



CodeWarrior Project Importer - Files Not Found

Some files or path contain white space, which can cause the build to fail.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

CONSOLE_EXTERNAL_RAM

C_4i_CF_Runtime.a
C_4i_CF_SZ_MSL.a

19. To locate the missing files, peform these steps.

a. Click **Back** to adjust the settings in the **Globals and Access Paths** pages so that the wizard can locate the missing files.

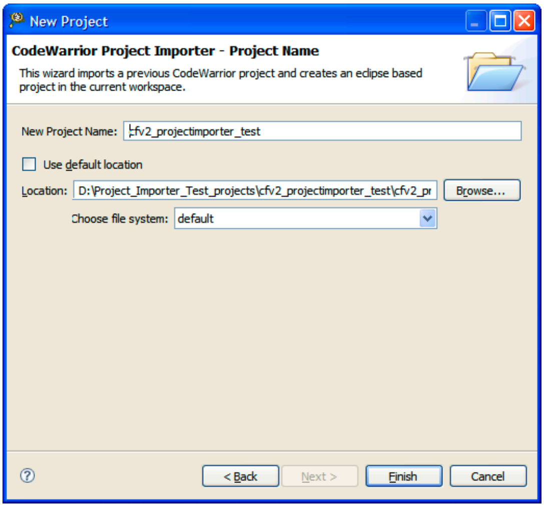
b. Repeat till you narrow down the number of missing files.

NOTE Some old files do not work with the <*target*> implementation, there will be some files missing.

20. Click **Next**.

The **CodeWarrior Project Importer - Project Name** page appears. This page lets you specify the name and select a location for the newly imported project.

Figure 2.146 CodeWarrior Project Importer — Project Name Page



CodeWarrior Project Importer - Project Name

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

New Project Name: cfv2_projectimporter_test

☐ Use default location

Location: D:\Project_Importer_Test_projects\cfv2_projectimporter_test\cfv2_pr

Choose file system: default

21. To specify a name and location to the imported project, perform these steps.

a. Enter a name for the converted Eclipse project, in the **New Project Name** text box. By default, the old project name is specified.

b. Check **Use default location** to save the project to the default Eclipse workspace. By default, the location of the project is the directory of the classic project and not the default Eclipse workspace.

TIP If you want to save the converted project to a location other than the default Eclipse workspace, click **Browse** and use the **Browse To Folder** dialog box to navigate to the desired directory.

22. Click **Finish**.

    The **CodeWarrior Classic Project Importer** wizard translates the classic CodeWarrior project and the new Eclipse project appears in the **CodeWarrior Project** view of the **Workbench** window.

NOTE Before debugging the new Eclipse project you might need to edit the build and launch configuration settings. For information on build properties, refer to the chapter Build Properties for Bareboard Projects.

Tutorial E: Porting Classic Power Architecture Project

The goal of this tutorial is to import a classic Power Architecture project to Eclipse.

NOTE Before starting the process ensure that the Power Architecture project you want to import has all of its files, such as the source, linker command, and settings file.

NOTE Screenshot in this tutorial will be available in an upcoming version.

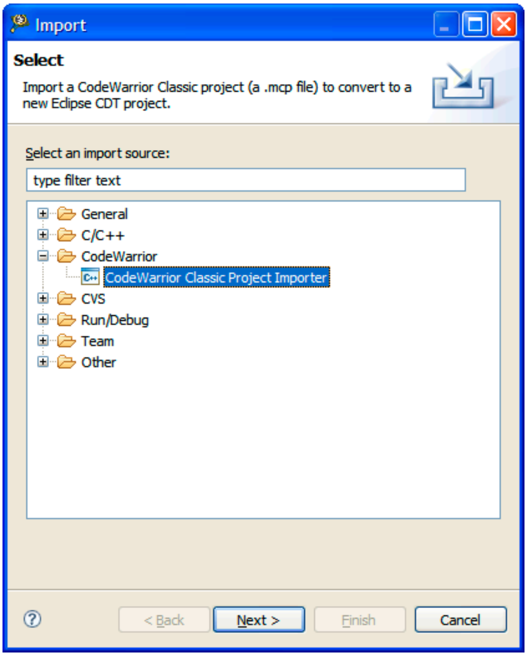
To port a classic Power Architecture project, perform these steps.

1. Select **File > Import** from the **Workbench** menu bar.

   The **Import** dialog box appears.

2. Expand the **CodeWarrior** tree control and select **CodeWarrior Classic Project Importer**.

Figure 2.147 Import Dialog Box



Import

Select

Import a CodeWarrior Classic project (a .mcp file) to convert to a new Eclipse CDT project.

Select an import source:

type filter text

General
C/C++
CodeWarrior
CodeWarrior Classic Project Importer
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

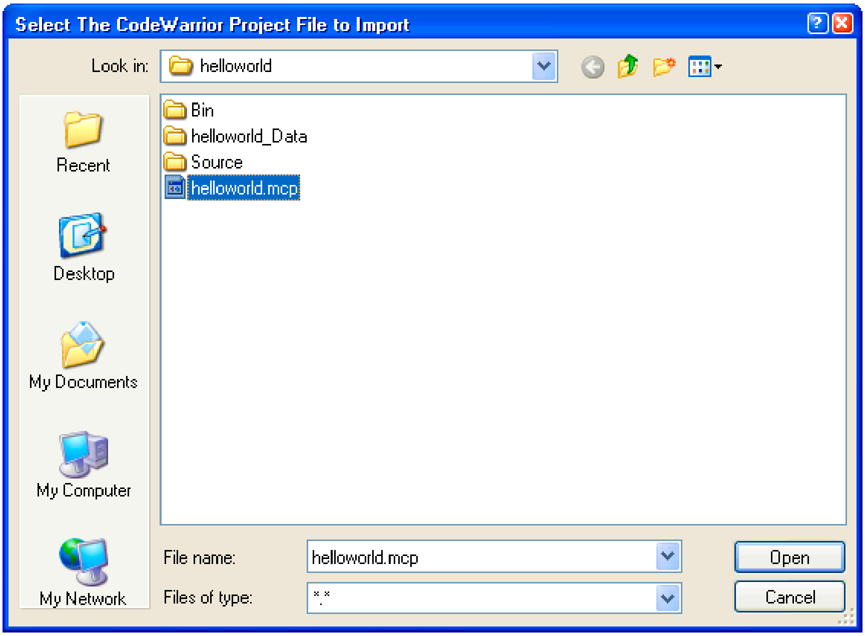
3. Click **Next**.

   The first page of the **CodeWarrior Classic Project Importer** wizard appears.

4. Enter the path and name of the classic CodeWarrior project file to import in the **Project file** text box. Alternatively, click **Browse** and use **Select The CodeWarrior Project File to Import** dialog box to select the project file to import. In this case, assume that the classic CodeWarrior project filename is `helloworld.mcp`.

TIP The project file has an extension of `.mcp`. Select the `.mcp` file.

Figure 2.148 Select The CodeWarrior Project File to Import Dialog Box

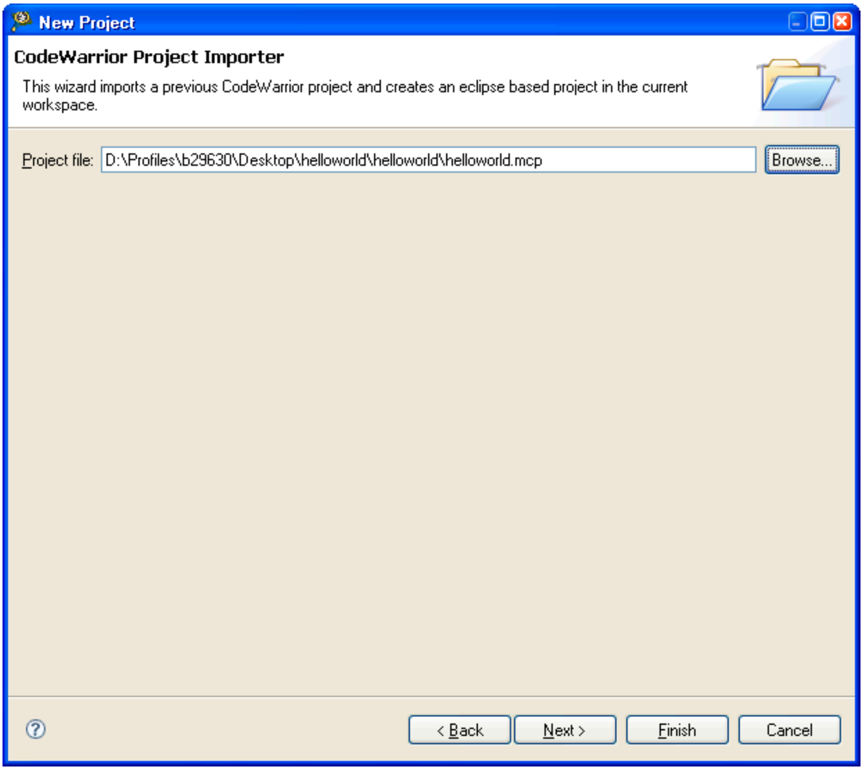

Select The CodeWarrior Project File to Import
Look in:
helloworld
Bin
helloworld_Data
Source
helloworld.mcp
Recent
Desktop
My Documents
My Computer
My Network
Open
Cancel

File name: helloworld.mcp

Files of type: *.*

The path of the project file to import appears in the **Project file** text box.

Figure 2.149 Path of CodeWarrior Project File to Import



New Project
CodeWarrior Project Importer

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

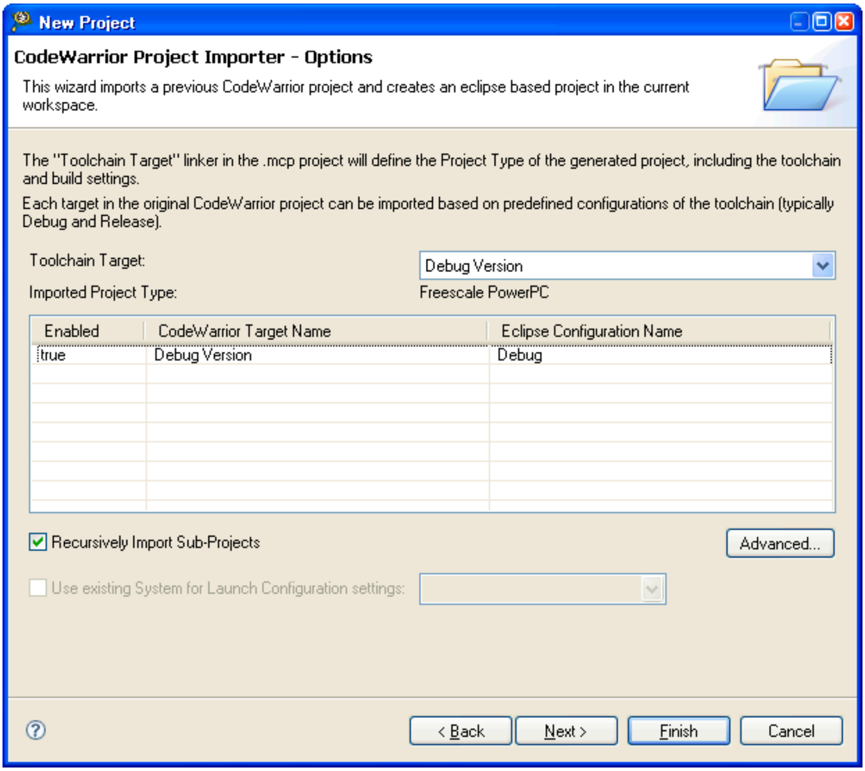
Project file: D:\Profiles\b29630\Desktop\helloworld\helloworld\helloworld.mcp

Browse...
< Back
Next >
Finish
Cancel

5. Click **Next**.

   The **Options** page of the **CodeWarrior Project Importer** wizard appears.

Figure 2.150 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: Debug Version

Imported Project Type: Freescale PowerPC

Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	Debug Version	Debug

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

6. Select the build target that uses the Power Architecture toolchain you want the generated Eclipse project to use, from the **Toolchain Target** list box.

NOTE The toolchain target linker in the classic project defines the project type of the generated Eclipse project, including the toolchain and build settings.

The build targets table displays all the targets discovered in the project file and is used to generate equivalent Eclipse build configurations.

7. You can import each build target in the classic CodeWarrior project based upon predefined configurations of the toolchain. For example: The Power Architecture project in the example lists these values:

- **Imported Project Type** = *Freescale PowerPC*
- **Enabled** = *true*
- **CodeWarrior Target Name** = *Debug Version*

- **Eclipse Configuration Name** = *Debug*

TIP To disable the generation of specific configurations, click a row in the build target table. In the **Edit Table Values** dialog box set **Enabled** to *false* and click **OK**.

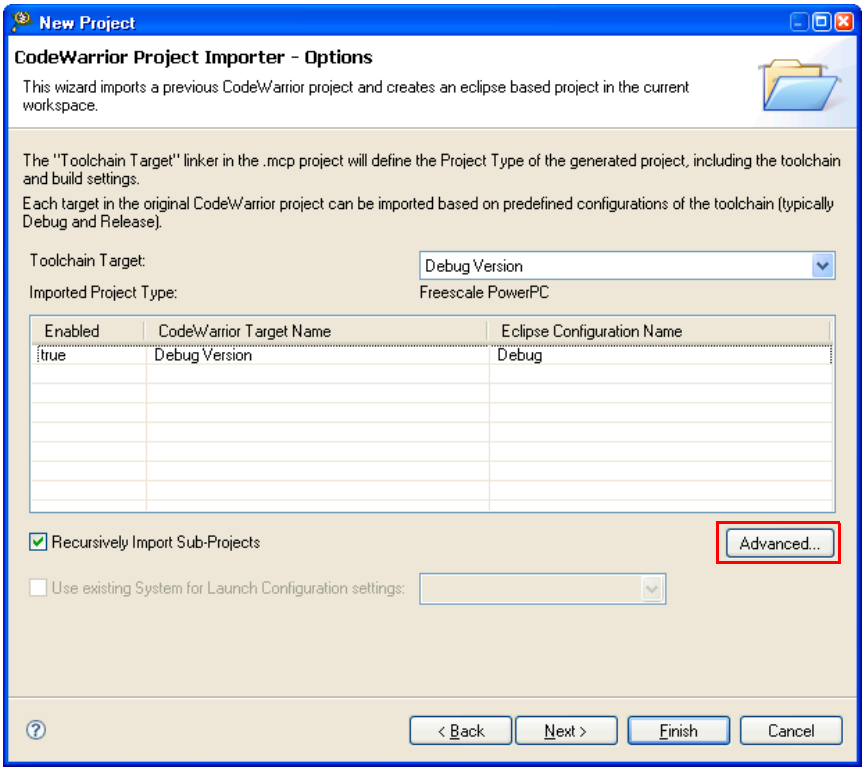
8. If you want to import sub-projects included in the classic CodeWarrior project, check the **Recursively Import Sub-Projects** checkbox.

   The **CodeWarrior Project Importer** wizard imports the sub-projects with the main project.

NOTE The **CodeWarrior Project Importer** wizard will copy only those files that are displayed in the **CodeWarrior Projects** view. The wizard will not import a file if it is not displayed or does not include project information.

9. Click **Advanced**.

Figure 2.151 CodeWarrior Project Importer — Options Page



CodeWarrior Project Importer - Options

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The "Toolchain Target" linker in the .mcp project will define the Project Type of the generated project, including the toolchain and build settings.

Each target in the original CodeWarrior project can be imported based on predefined configurations of the toolchain (typically Debug and Release).

Toolchain Target: Debug Version

Imported Project Type: Freescale PowerPC

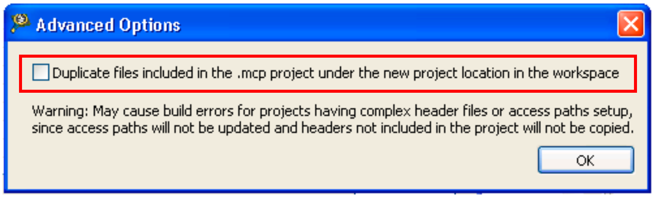
Enabled	CodeWarrior Target Name	Eclipse Configuration Name
true	Debug Version	Debug

☑ Recursively Import Sub-Projects

☐ Use existing System for Launch Configuration settings:

The **Advanced Options** dialog box appears.

Figure 2.152 Advanced Options



☐ Duplicate files included in the .mcp project under the new project location in the workspace

Warning: May cause build errors for projects having complex header files or access paths setup, since access paths will not be updated and headers not included in the project will not be copied.

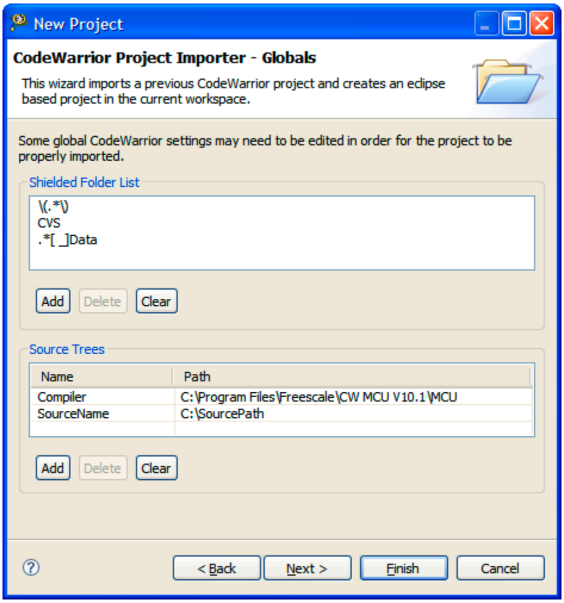
WARNING! Checking the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox may cause build errors.

10. Check the **Duplicate files included in the .mcp project under the new project location in the workspace** checkbox.

11. Click **OK** to close the dialog box.
12. Click **Next**.

    The **Globals** page of the **CodeWarrior Project Importer** wizard appears. This page lets you edit global settings that can effect how the project's build options are imported.

Figure 2.153 CodeWarrior Project Importer — Globals Page



CodeWarrior Project Importer - Globals

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Some global CodeWarrior settings may need to be edited in order for the project to be properly imported.

Shielded Folder List

```
\(.*\)
CVS
.*[ _]Data
```

Source Trees

Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

Table 2.35 lists the options on the **CodeWarrior Project Importer — Globals** page.

Table 2.35 CodeWarrior Project Importer — Globals Page Options

Options	Description
Shielded Folder List	Previous versions of the CodeWarrior tools let you conceal the contents of folders from the IDE's search operations during a build. This was done by placing special characters in the directory name. For example, sample code was concealed in a (CodeWarrior Examples) folder. The **Shielded Folder List** table lists these options: • \(.*\) • CVS • .*[_]Data You use the **Add, Delete**, and **Clear** buttons to modify the information in this list. Table 2.36 lists these buttons with their descriptions.
Source Trees	Specifies the location of the source trees. If an access path is defined relative to a source tree, the source tree should be listed in this table. The **{Project}** source tree is defined automatically. The **Source Trees** table lists these options: • Name — Lists the source name. For example: Compiler. • Compiler — Lists the path source name. For example: `<CW Install>/MCU`. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.36 lists these buttons with their descriptions.

Table 2.36 CodeWarrior Project Importer — Globals Page Buttons

Button	Description
Add	Add a new entry to the list.
Delete	Deletes the selected item.
Clear	Clears the entire list.

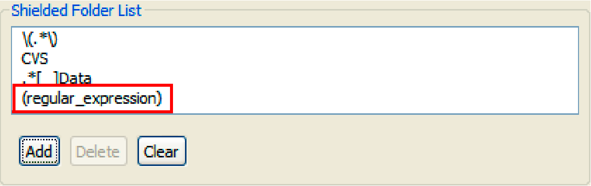
a. Click **Add**.

(regular_expression) appears in the shielded folder list.

b. Double-click *(regular_expression)* and type the required expression.

The new expression appears in shielded folder list.

Figure 2.154 Shielded Folder List



```
\(.*\)
CVS
.*[ ]Data
(regular_expression)
```

13. To delete an existing expression from the **Shielded Folder List** table, select the expression and click **Delete**.

    The selected expression is deleted from the shielded folder list.

14. To remove all the existing expressions from the **Shielded Folder List** table, click **Clear**.

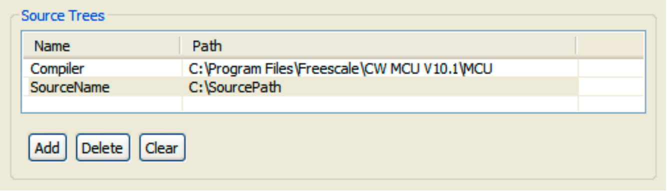
    All the expressions are deleted from the shielded folder list.

15. To add a new source to the **Source Trees** table, perform these steps.

    a. Click **Add**.

    SourceName appears in the source trees list.

Figure 2.155 Sources Trees

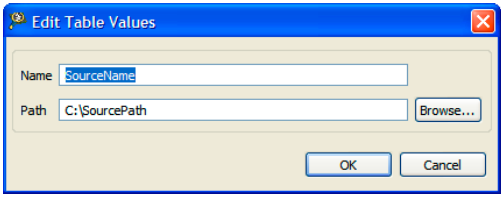


Name	Path
Compiler	C:\Program Files\Freescale\CW MCU V10.1\MCU
SourceName	C:\SourcePath

b. Double-click *SourceName.*

The **Edit Table Values** dialog box appears.

Figure 2.156 Edit Table Values



Name: SourceName

Path: C:\SourcePath

c. In the **Name** text box, enter the source name.

d. In the **Path** text box, enter the path of the new source. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

e. Click **OK**.

The new source appears in **Source Trees** list.

16. Click **Next**.

The **CodeWarrior Project Importer - Access Paths** page appears.

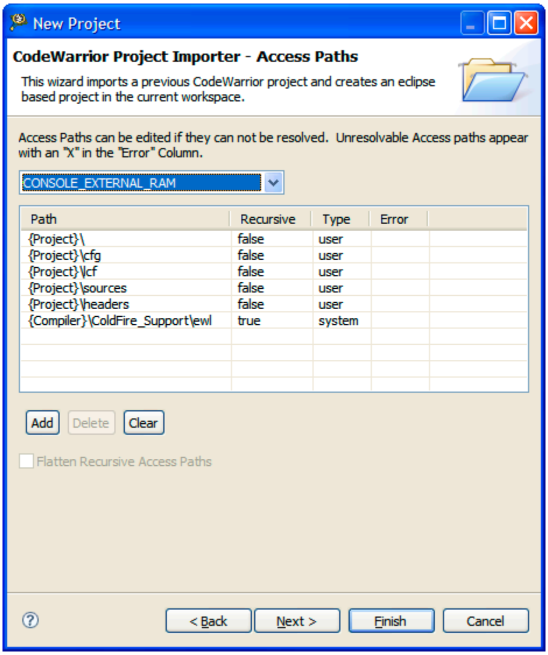
NOTE Access paths are directory paths the CodeWarrior tools use to search for libraries, runtime support files, and other object files.

Table 2.37 lists the options on the **CodeWarrior Project Importer — Access Paths** page.

Table 2.37 CodeWarrior Project Importer — Access Paths Page

Option	Description
Build Target List Box	Lets you select the build target whose access paths you want to modify. For example: `CONSOLE_EXTERNAL_RAM.`
Access Path Table	Lists the access paths used by the build target selected in the **Build Target** list box. Each row lists: • **Path** — Directory path. For example: The path is `{Compiler}\PA_Support\ewl.` • **Recursive** — Whether the path is searched recursively. For example: `false` or `true.` • **Type** — Specify of path to be searched. For example: `user` or `system.` • **Error** — Unresolved access paths, marked as “X”, if any. You use the **Add**, **Delete**, and **Clear** buttons to modify the information in this list. Table 2.38 lists these buttons with their descriptions.
Flatten Recursive Access Paths Checkbox	When checked, the **CodeWarrior Project Importer** wizard automatically generates separate include paths for each subdirectory that is a part of the recursive path. This option is set by default as most compilers do not support recursive include paths passed by the command line.

Figure 2.157 CodeWarrior Project Importer — Access Paths Page



CodeWarrior Project Importer - Access Paths

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Access Paths can be edited if they can not be resolved. Unresolvable Access paths appear with an "X" in the "Error" Column.

CONSOLE_EXTERNAL_RAM

Path	Recursive	Type	Error
{Project}\	false	user	
{Project}\cfg	false	user	
{Project}\lcf	false	user	
{Project}\sources	false	user	
{Project}\headers	false	user	
{Compiler}\ColdFire_Support\ewl	true	system	

Table 2.38 CodeWarrior Project Importer — Access Paths Page Buttons

Button	Description
Add	Add a new directory path to the list.
Delete	Deletes the selected directory path from the list.
Clear	Clears the entire list.

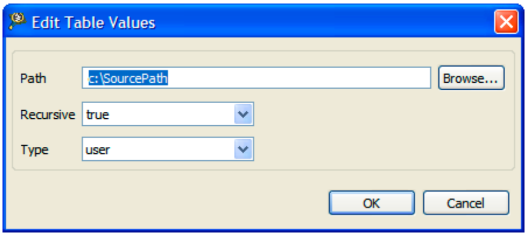
17. To add a new directory path to list, perform these steps.
    a. Click **Add**.

       (C:\SourcePath) appears in the **Path** list.

    b. Click *(C:\SourcePath)*.

       The **Edit Table Values** dialog box appears.

Figure 2.158 Edit Table Values


Edit Table Values
Path
c:\SourcePath
Browse...
Recursive
true
Type
user
OK
Cancel

c. In the **Path** text box, enter the directory path. Alternatively, click **Browse** and use the **Browse For Folder** dialog box and navigate to the required source.

d. From the **Recursive** list box, select `false` or `true`.

e. From the **Type** list box, select `user` or `system`.

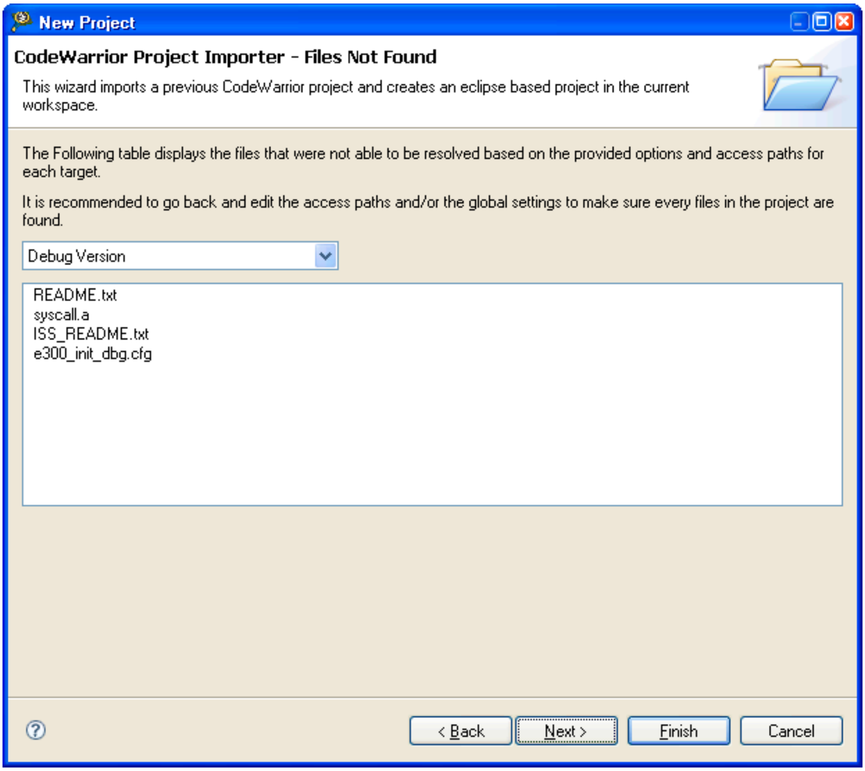
f. Click **OK**.

The new access path appears in table.

18. Click **Next**.

The **CodeWarrior Project Importer — Files Not Found** page appears. This page displays the project files that the wizard could not locate. You can use the Build Target list box to select another build target and view the missing files.

Figure 2.159 CodeWarrior Project Importer — Files Not Found Page



CodeWarrior Project Importer - Files Not Found

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

The Following table displays the files that were not able to be resolved based on the provided options and access paths for each target.

It is recommended to go back and edit the access paths and/or the global settings to make sure every files in the project are found.

Debug Version

README.txt
syscall.a
ISS_README.txt
e300_init_dbg.cfg

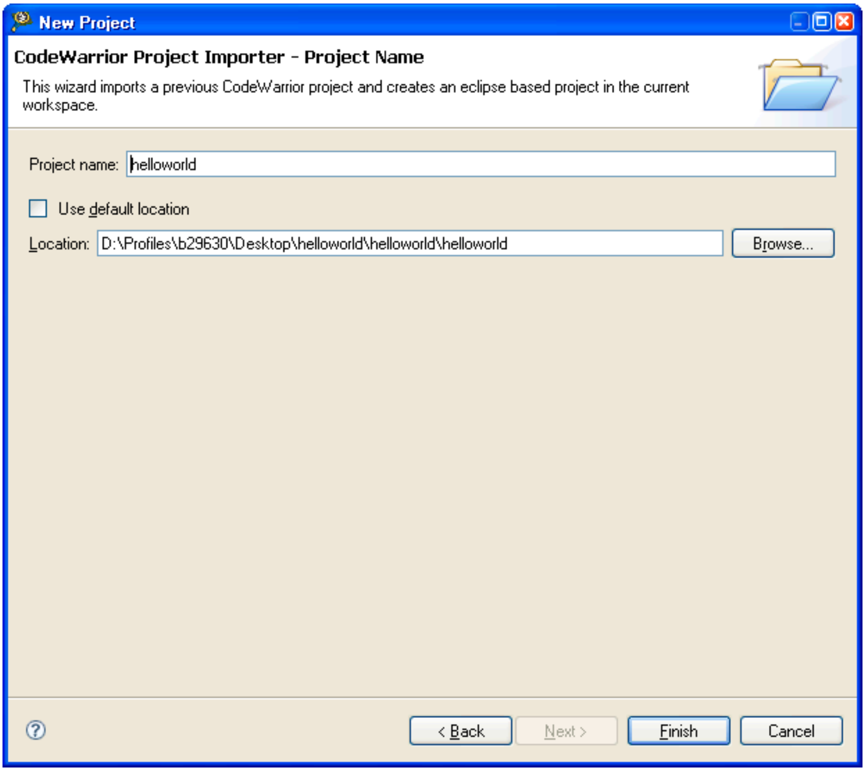
19. To locate the missing files, peform these steps.
    a. Click **Back** to adjust the settings in the **Globals and Access Paths** pages so that the wizard can locate the missing files.
    b. Repeat till you narrow down the number of missing files.

NOTE Some old files do not work with the <*target*> implementation, there will be some files missing.

20. Click **Next**.

    The **CodeWarrior Project Importer - Project Name** page appears. This page lets you specify the name and select a location for the newly imported project.

Figure 2.160 CodeWarrior Project Importer — Project Name Page



New Project

CodeWarrior Project Importer - Project Name

This wizard imports a previous CodeWarrior project and creates an eclipse based project in the current workspace.

Project name: helloworld

☐ Use default location

Location: D:\Profiles\b29630\Desktop\helloworld\helloworld\helloworld

21. To specify a name and location to the imported project, perform these steps.
    a. Enter a name for the converted Eclipse project, in the **New Project Name** text box. By default, the old project name is specified.
    b. Check **Use default location** to save the project to the default Eclipse workspace. By default, the location of the project is the directory of the classic project and not the default Eclipse workspace.

TIP If you want to save the converted project to a location other than the default Eclipse workspace, click **Browse** and use the **Browse To Folder** dialog box to navigate to the desired directory.

22. Click **Finish**.

The **CodeWarrior Classic Project Importer** wizard translates the classic CodeWarrior project and the new Eclipse project appears in the **CodeWarrior Project** view of the **Workbench** window.

NOTE Before debugging the new Eclipse project you might need to edit the build and launch configuration settings. For information on build properties, refer to the chapter Build Properties for Bareboard Projects.

3

Build Properties for Bareboard Projects

This chapter explains build properties for a Microcontrollers project. The Microcontrollers **New Bareboard Project** wizard uses the information it gathers from you to set up the project's build and launch configurations.

A project's *build configuration* contains information on the tool settings used to make the program. For example, it describes the compiler and linker settings, and the files involved, such as source and libraries.

A project's *launch configuration* describes how the IDE starts the program, such as whether it executes by itself on a target, or under debugger control. Launch configurations also specify the core the program executes on (if the target processor has multiple cores). They also specify the connection interface and communications protocol that the debugger uses to control the environment that the program executes in.

NOTE The settings of the CodeWarrior IDE's build and launch configuration correspond to an object called a *target* made by the classic CodeWarrior IDE.

When the **New Bareboard Project** wizard completes its process, it generates launch configurations with names that follow the pattern ***projectname*** - ***configtype*** - ***targettype***, where:

- ***projectname*** represents the name of the project
- ***configtype*** represents the project's name, which usually describes the build configuration
- ***targettype*** represents the type of target software or hardware on which the launch configuration acts

For each launch configuration, you can specify build properties, such as:

- additional libraries to use for building code
- behavior of the compilers, linkers, assemblers, and other build-related tools
- specific build properties, such as the byte ordering of the generated code

The topics in this chapter are:

- Changing Build Properties
- Restoring Build Properties

- Defining C/C++ Build Settings and Behavior
- Build Properties for HCS08
- Build Properties for RS08
- Build Properties for ColdFire
- Build Properties for Power Architecture
- Build Properties for ARM

Changing Build Properties

The **New Bareboard Project** wizard creates a set of build properties for the project. You can modify these build properties to better suit your needs.

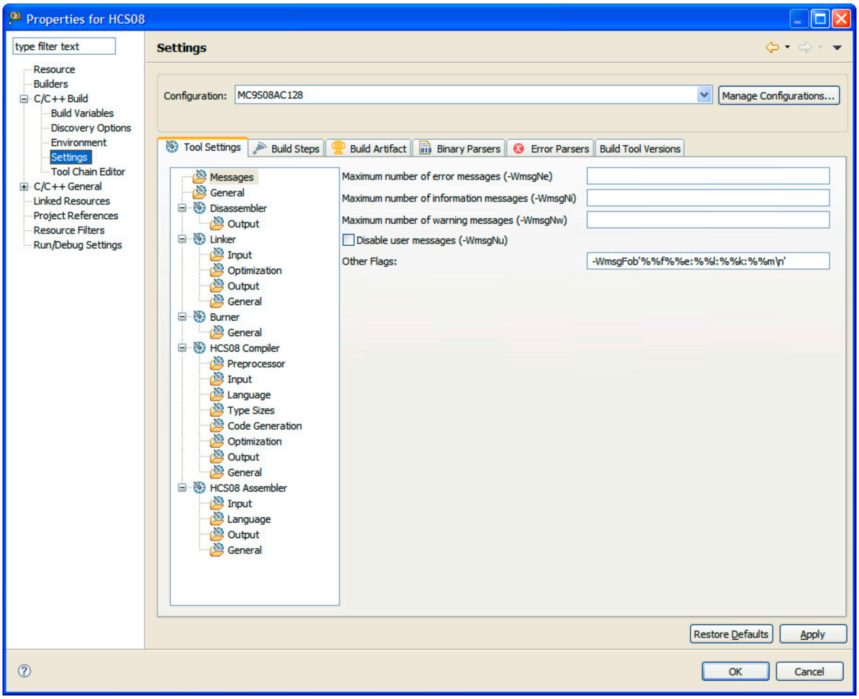
Perform these steps to change build properties:

1. Start the IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build properties.
3. Select **Project > Properties**.

   The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Expand the **C/C++ Build** property.
5. Select **Settings**.

   The **Properties** window shows the corresponding build properties as in Figure 3.1.

Figure 3.1 Properties for *<Project>* Window



Properties for HCS08
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
General
Disassembler
Output
Linker
Input
Optimization
Output
General
Burner
General
HCS08 Compiler
Preprocessor
Input
Language
Type Sizes
Code Generation
Optimization
Output
General
HCS08 Assembler
Input
Language
Output
General
Restore Defaults
Apply
OK
Cancel

Configuration: MC9S08AC128

Maximum number of error messages (-WmsgNe)

Maximum number of information messages (-WmsgNi)

Maximum number of warning messages (-WmsgNw)

☐ Disable user messages (-WmsgNu)

Other Flags: -WmsgFob'%%f%%e:%%l:%%k:%%m\n'

6. Use the **Configuration** drop-down list to specify the launch configuration for which you want to modify the build properties.
7. Click the **Tool Settings** tab.

   The corresponding page appears.
8. From the list of tools on the **Tool Settings** page, select the tool for which you want to modify properties.
9. Change the settings that appear in the page.
10. Click **Apply**.

    The IDE saves your new settings.

You can select other tool pages and modify their settings. When you finish, click **OK** to save your changes and close the **Properties** window.

Restoring Build Properties

If you modify a build configuration that the new project wizard generates, you can restore that configuration to its default state. You might want to restore the build properties in order to have a factory-default configuration, or to revert to a last-known working build configuration. To undo your modifications to build properties, click the **Restore Defaults** button at the bottom of the **Properties** window.

This changes the values of the options to the absolute default of the toolchain. By default, the toolchain options are blank.

For example, when a HCS08 project is created the **Linker > Input** panel has some values set for the **Parameter File** and **Libraries** options, which are specific to the project. Clicking the **Restore Defaults** button defaults the values of the **Parameter File** and **Libraries** options to the blank state of the toolchain.

Defining C/C++ Build Settings and Behavior

The **C/C++ Build** page includes all builder-specific property pages. You can use the **C/C++ Build** page to:

- Define Build Settings
- Define Build Behavior

NOTE Modifying settings such as the Generate makefiles automatically option, might enable or disable some parameters in some situations and change the availability of other property pages.

Define Build Settings

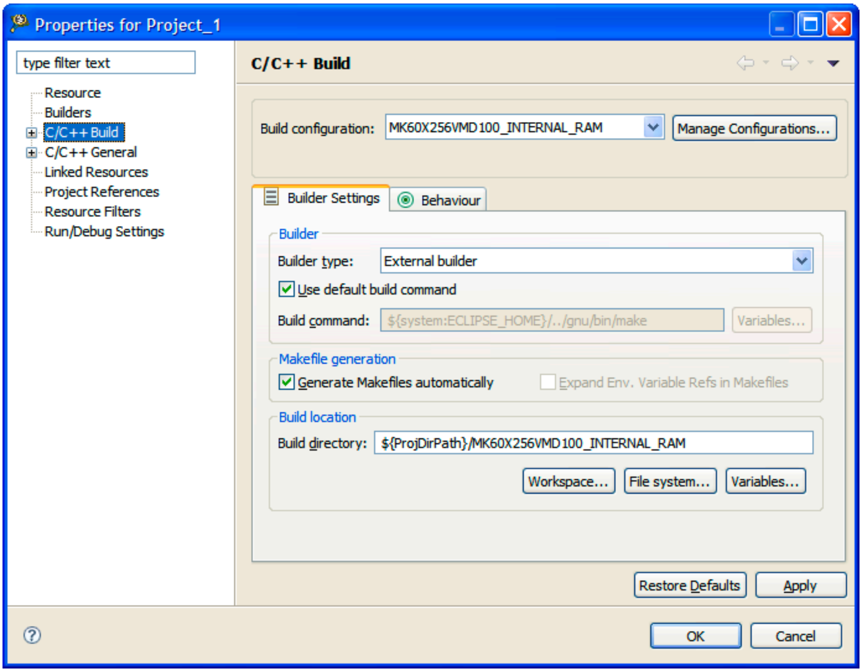
To define build settings, perform these steps.

1. Start the IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build settings.
3. Select **Project > Properties**.

   The **Properties for <project>** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Select **C/C++ Build**.

   The **C/C++ Build** page appears.

Figure 3.2 C/C++ Build Page — Builder Settings


Properties for Project_1
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
C/C++ Build
Manage Configurations...
Builder Settings
Behaviour
Builder
Variables...
Makefile generation
Build location
Workspace...
File system...
Variables...
Restore Defaults
Apply
OK
Cancel

Build configuration: MK60X256VMD100_INTERNAL_RAM

Builder type: External builder

☑ Use default build command

Build command: ${system:ECLIPSE_HOME}/../gnu/bin/make

☑ Generate Makefiles automatically

☐ Expand Env. Variable Refs in Makefiles

Build directory: ${ProjDirPath}/MK60X256VMD100_INTERNAL_RAM

5. Click the **Builder Settings** tab.

   The builder settings for the selected build configuration appears. Table 3.1 describes the builder settings options.

Table 3.1 Builder Settings Options

Group	Option	Description
Build Configuration	Configuration	Specifies the type of configurations for the selected project.
Build Configuration	Manage configurations	Click to open the **Manage Configurations** dialog box that lets you can set configurations based on the specified toolchains of the selected project. You can also create new configurations, rename an existing configuration, or remove ones that are no longer required.

Table 3.1 Builder Settings Options (*continued*)

Group	Option	Description
Builder	Builder type	Specifies the type of builder to use: • Internal builder — Builds C/C++ programs using a compiler that implements the C/C++ Language Specifications. • External builder — External tools let you configure and run programs and Ant buildfiles using the Workbench, which can saved and run at a later time to perform a build).
Builder	Use default build command	Check to indicate that you want to use the default make command. Clear when you want to use a new make command. This option is only available when the Build type option is set to External.
Builder	Build command	Specifies the default command used to start the build utility for your specific toolchain. Use this field if you want to use a build utility other than the default make command.
Builder	Variables	Click to open the **Select build variable** dialog box and add the desired environment variables and custom variables to the build command.
Makefile generation	Generate Makefiles automatically	Check to enable Eclipse change between two different CDT modes: it either uses the customer's makefile for the build, if one exists, or it generate makefiles for the user.
Makefile generation	Expand Env. Variable Refs in Makefiles	Check to defines whether environment variables should be expanded in makefile.
Build location	Build directory	Specifies the location where the build operation takes place. This location will contain the generated artifacts from the build process. This option appears disabled when the Generate Makefiles automatically option is enabled.

Table 3.1 Builder Settings Options (*continued*)

Group	Option	Description
Build location	Workspace	Click to open the **Folder Selection** dialog box and select a workspace location for the project. This is the directory that will contain the plug-ins and features to build, including any generated artifacts.
Build location	File system	Click to open the **Browse For Folder** dialog box and select a folder.
Build location	Variables	Click to open the **Select build variable** dialog box and select a variable to specify as an argument for the build directory, or create and configure simple build variables which you can reference in build configurations that support variables.

6. Make the desired changes and click **OK**.

   The **Properties for <project>** window will close.

Define Build Behavior

To define build behavior, perform these steps.

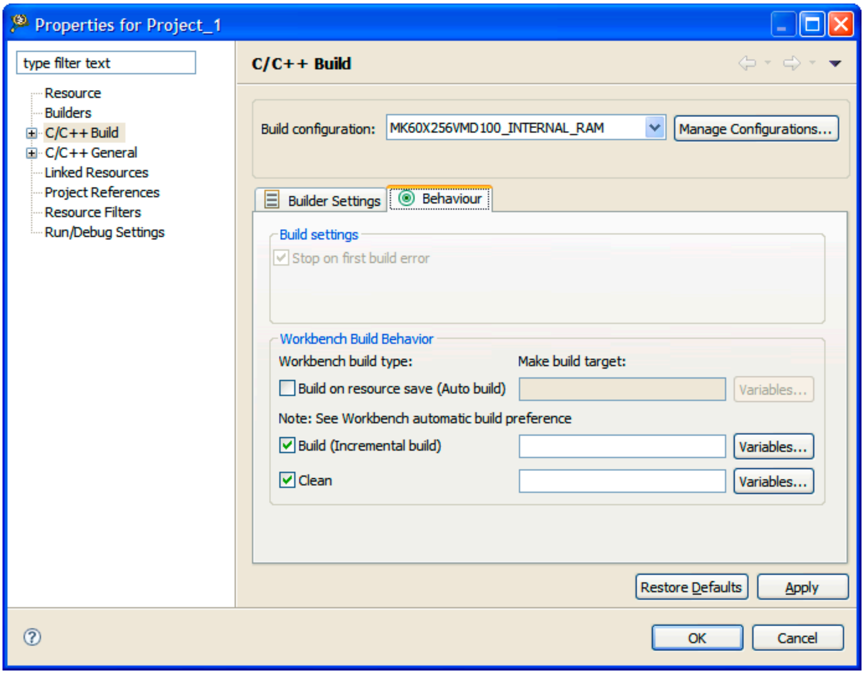
1. Start the IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build settings.
3. Select **Project > Properties**.

   The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Select **C/C++ Build**.

   The **C/C++ Build** page appears.
5. Click the **Behaviour** tab.

   The behavior settings for the selected build configuration appears.

Figure 3.3 C/C++ Build Page — Behaviour (External Builder)


Properties for Project_1
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
C/C++ Build
Manage Configurations...
Builder Settings
Behaviour
Build settings
Workbench Build Behavior
Workbench build type:
Make build target:
Variables...
Variables...
Variables...
Restore Defaults
Apply
OK
Cancel

Build configuration: MK60X256VMD100_INTERNAL_RAM

☑ Stop on first build error

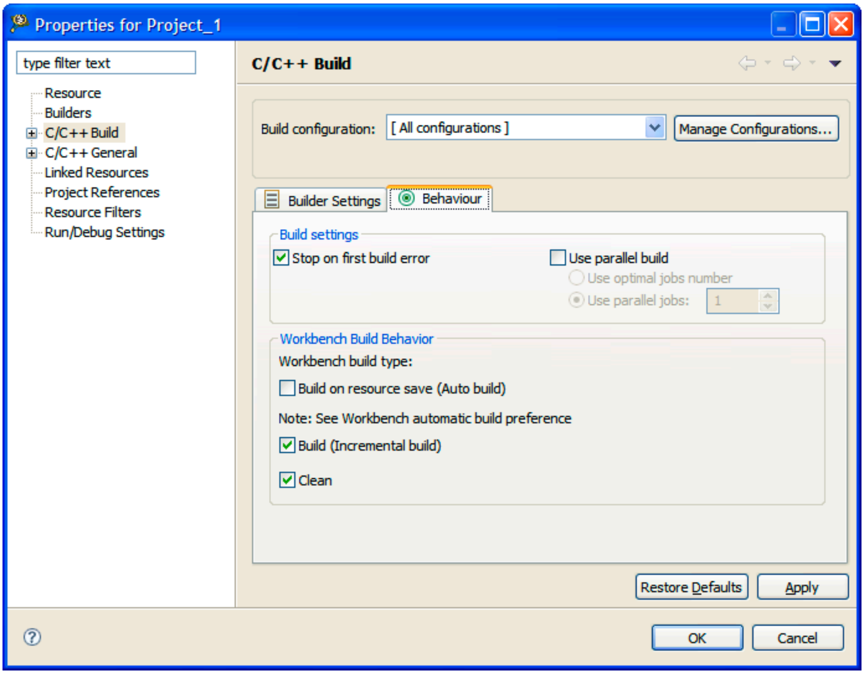
☐ Build on resource save (Auto build)

Note: See Workbench automatic build preference

☑ Build (Incremental build)

☑ Clean

Figure 3.4 C/C++ Build Page — Behaviour (Internal Builder)


Properties for Project_1
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
C/C++ Build
Manage Configurations...
Builder Settings
Behaviour
Restore Defaults
Apply
OK
Cancel

Build configuration: [All configurations]

Build settings

☑ Stop on first build error

☐ Use parallel build

○ Use optimal jobs number

● Use parallel jobs: 1

Workbench Build Behavior

Workbench build type:

☐ Build on resource save (Auto build)

Note: See Workbench automatic build preference

☑ Build (Incremental build)

☑ Clean

Table 3.2 describes the builder settings options.

Table 3.2 Behavior Options

Group	Option	Description
Build settings	Stop on first build error	Check to stops building when Eclipse encounters an error. Clearing this option is helpful for building large projects as it enables make to continue making other independent rules even when one rule fails.
Build settings	Use parallel build	Check to activate the generation of parallel builds. However, you need to determine the number of parallel jobs to perform: • Use optimal jobs number — Lets the system determine the optimal number of parallel jobs to perform. • Use parallel jobs — Lets you specify the maximum number of parallel jobs to perform.
Workbench Build Behavior	Workbench build behavior type	Specifies the builder settings when instructed to build, rebuild, and clean.
Workbench Build Behavior	Build on resource save (Auto build)	Check to build your project whenever resources are saved. By default, this option is selected and builds occur automatically each time resources are modified. Clear if you do want that the build occurs only manually using a menu item.
Workbench Build Behavior	Build (Incremental Build)	Defines what the standard builder will call when an incremental build is performed.
Workbench Build Behavior	Variables	Click to open the **Select build variable** dialog box and add variables to the make build target command.
Workbench Build Behavior	Clean	Defines what the standard builder calls when a clean is performed. The make clean is defined in the makefile.
Workbench Build Behavior	Variables	Click to open the **Select build variable** dialog box and add variables to the make build target command.

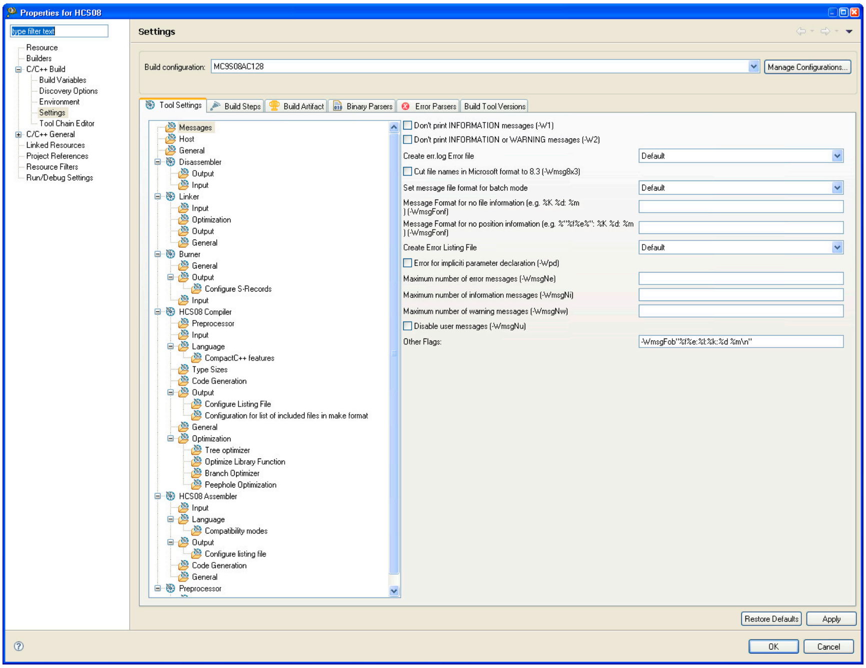
6. Make the desired changes and click **OK**.

   The **Properties for <project>** window will close.

Build Properties for HCS08

The **Properties for *<project>*** window shows the corresponding build properties for an HCS08 project (Figure 3.5).

Figure 3.5 Build Properties — HCS08


Properties for HCS08
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Manage Configurations...
Restore Defaults
Apply
OK
Cancel

Build configuration: MC9S08AC128

☐ Don't print INFORMATION messages (-W1)
☐ Don't print INFORMATION or WARNING messages (-W2)
Create err.log Error file: Default
☐ Cut file names in Microsoft format to 8.3 (-Wmsg8x3)
Set message file format for batch mode: Default
Message Format for no file information (e.g. %K %d: %m) (-WmsgFonf):
Message Format for no position information (e.g. %"%f%e%": %K %d: %m) (-WmsgFonf):
Create Error Listing File: Default
☐ Error for impliciti parameter declaration (-Wpd)
Maximum number of error messages (-WmsgNe):
Maximum number of information messages (-WmsgNi):
Maximum number of warning messages (-WmsgNw):
☐ Disable user messages (-WmsgNu)
Other Flags: -WmsgFob"%f%e:%l:%k:%d %m\n"

Table 3.3 lists the build properties specific to developing software for HCS08.

The properties that you specify in the **Tool Settings** panels apply to the selected build tool on the **Tool Settings** page of the **Properties for *<project>*** window.

Table 3.3 Build Properties for HCS08

Build Tool	Build Properties Panels
Messages	Messages
Host	Host
General	General
Disassembler	Disassembler > Output
	Disassembler > Input
Linker	Linker > Input
	Linker > Optimization
	Linker > Output
	Linker > General
Burner	Burner > General
	Burner > Output • Burner > Output > Configure S-Record
	Burner > Input

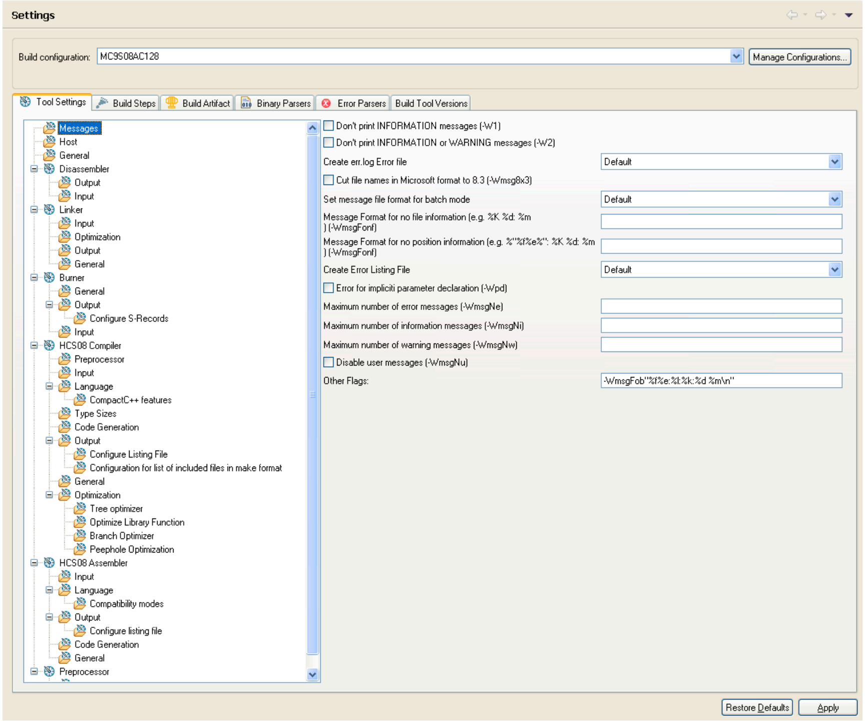
Table 3.3 Build Properties for HCS08 (*continued*)

Build Tool	Build Properties Panels
HCS08 Compiler	HCS08 Compiler > Preprocessor
	HCS08 Compiler > Input
	HCS08 Compiler > Language • HCS08 Compiler > Language > CompactC++ features
	HCS08 Compiler > Type Sizes
	HCS08 Compiler > Code Generation
	HCS08 Compiler > Output • HCS08 Compiler > Output > Configure Listing File • HCS08 Compiler > Output > Configuration for list of included files in make format
	HCS08 Assembler > General
	HCS08 Compiler > Optimization • HCS08 Compiler > Optimization > Tree optimizer • HCS08 Compiler > Optimization > Optimize Library Function • HCS08 Compiler > Optimization > Branch Optimizer • HCS08 Compiler > Optimization > Peephole Optimization
HCS08 Assembler	HCS08 Assembler > Input
	HCS08 Assembler > Language • HCS08 Assembler > Language > Compatibility modes
	HCS08 Assembler > Output • HCS08 Assembler > Output > Configure listing file
	HCS08 Assembler > Code Generation
	HCS08 Assembler > General
Preprocessor	Preprocessor > Preprocessor Settings

Messages

Use this panel to specify whether to generate symbolic information for debugging the build target (Figure 3.6).

Figure 3.6 Tool Settings — Messages


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Don't print INFORMATION messages (-W1)
☐ Don't print INFORMATION or WARNING messages (-W2)
Create err.log Error file: Default
☐ Cut file names in Microsoft format to 8.3 (-Wmsg8x3)
Set message file format for batch mode: Default
Message Format for no file information (e.g. %K %d: %m) (-WmsgFonf):
Message Format for no position information (e.g. %"%f%e%": %K %d: %m) (-WmsgFonf):
Create Error Listing File: Default
☐ Error for impliciti parameter declaration (-Wpd)
Maximum number of error messages (-WmsgNe):
Maximum number of information messages (-WmsgNi):
Maximum number of warning messages (-WmsgNw):
☐ Disable user messages (-WmsgNu)
Other Flags: -WmsgFob"%f%e:%l:%k:%d %m\n"

Table 3.4 lists and describes the message options.

Table 3.4 Tool Settings — Messages Options

Option	Description
Don’t print INFORMATION messages (-W1)	Inhibits information message reporting. Only warning and error messages are generated.
Don’t print INFORMATION or WARNING messages (-W2)	Suppresses all messages of type INFORMATION and WARNING. Only ERROR messages are generated.

Table 3.4 Tool Settings — Messages Options (*continued*)

Option	Description
Create err.log Error file	Using this option, the Compiler uses a return code to report errors back to the tools. When errors occur, 16-bit window environments use err.log files, containing a list of error numbers, to report the errors. If no errors occur, the 16-bit window environments delete the err.log file.
Cut file names to Microsoft format to 8.3 (-Wmsg8x3)	Some editors (early versions of WinEdit) expect the filename in Microsoft message format (8.3 format). That means the filename can have up to eight characters and no more than a three-character extension. Longer filenames are possible when you use Win95 or WinNT. This option truncates the filename to the 8.3 format.
Set message file format for batch mode	Use this option to start the Compiler with additional arguments (for example, files and Compiler options). If you start the Compiler with arguments (for example, from the Make Tool or with the ‘%f’ argument from the CodeWright IDE), the Compiler compiles the files in a batch mode. No Compiler window is visible and the Compiler terminates after job completion.
Message Format for no file information (e.g. %K %d: %m)(-WmsgFonf)	If there is no file information available for a message, then <string> defines the message format string to use.
Message Format for no positioning information (%”%f%e%”:%K %d: %m)(-WmsgFonf)	If there is no position information available for a message, then <string> defines the message format string to use.
Create Error Listing File	This option controls whether the Compiler creates an error listing file. The error listing file contains a list of all messages and errors that occur during processing.

Table 3.4 Tool Settings — Messages Options (*continued*)

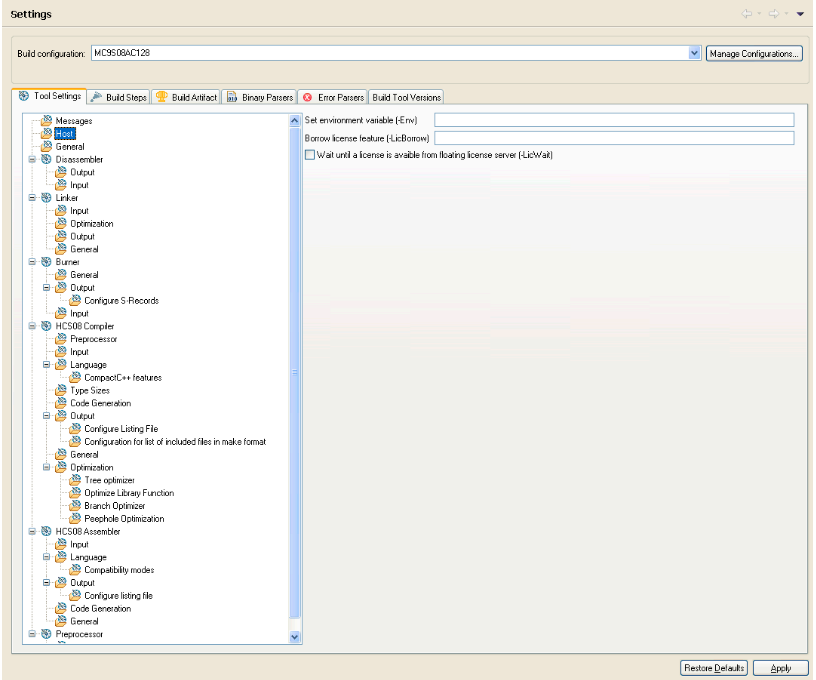
Option	Description
Error for Impliciti parameter declaration (-Wpd)	This option prompts the Compiler to issue an ERROR message instead of a WARNING message when the Compiler encounters an implicit declaration. This occurs if the Compiler does not have a prototype for the called function. This option helps prevent parameter-passing errors, which can only be detected at runtime. It requires prototyping each called function before use. Correct ANSI behavior assumes that parameters are correct for the stated call. This option is the same as using -`WmsgSe1801`.
Maximum number of error messages (-WmsgNe)	Specify the number of errors allowed until the application stops processing.
Maximum number of information messages (-WmsgNi)	Specify the maximum number of information messages allowed.
Maximum number of warning messages (-WmsgNw)	Specify the maximum number of warnings allowed.
Disable user messages (-WmsgNu)	Check to disable user messages and allow only the normal message categories (WARNING, INFORMATION, ERROR, or FATAL); reduces the number of messages, and simplifies the error parsing of other tools.
Other Flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI. Default value is `-WmsgFob"%%f%%e:%%l: %%k: %%m\n"`

Host

Use this panel to specify the host settings of the HCS08.

Figure 3.7 shows the Host settings.

Figure 3.7 Tool Settings — Host


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Set environment variable (-Env)

Borrow license feature (-LicBorrow)

☐ Wait until a license is avaible from floating license server (-LicWait)

Table 3.6 lists and describes the memory model options for HCS08.

Table 3.5 Tool Settings — Host

Option	Description
Set environment variable (-Env)	This option sets an environment variable. Use this environment variable in the maker, or use to overwrite system environment variables.

Table 3.5 Tool Settings — Host (*continued*)

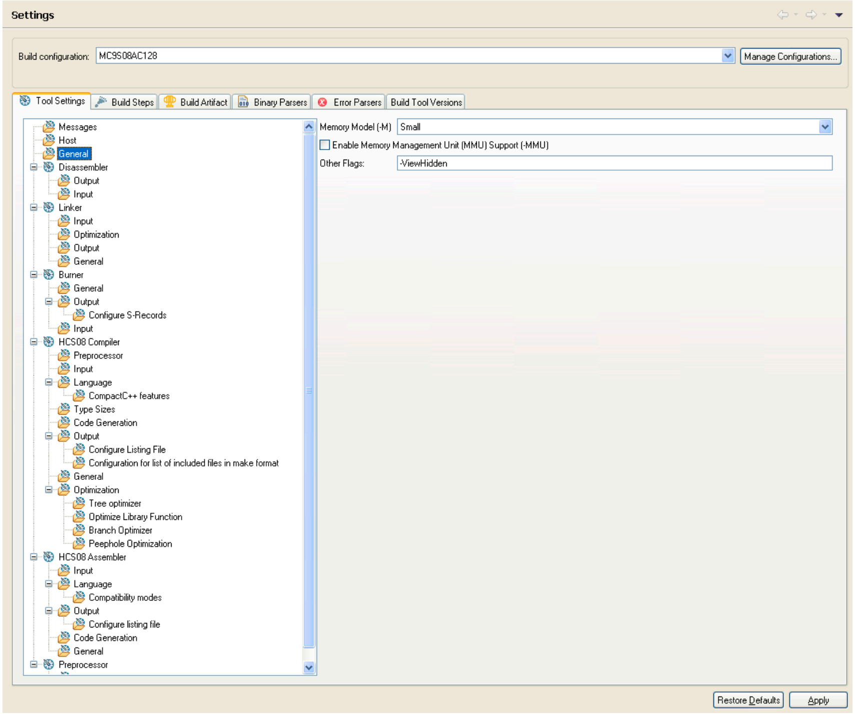
Option	Description
Borrow license feature (-LicBorrow)	This option allows you to borrow a license feature until a given date or time. Borrowing allows you to use a floating license even if disconnected from the floating license server.
Wait until a license is available from floating license server (-LicWait)	By default, if a license is not available from the floating license server, then the application will immediately return. With -LicWait set, the application will wait (blocking) until a license is available from the floating license server.

General

Use this panel to specify the memory model that the architecture uses. The build tools (compiler, linker, and assembler) use the properties that you specify.

Figure 3.8 shows the **General** settings.

Figure 3.8 Tool Settings — General


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Memory Model (-M) Small

☐ Enable Memory Management Unit (MMU) Support (-MMU)

Other Flags: -ViewHidden

Table 3.6 lists and describes the memory model options for HCS08.

Table 3.6 Tool Settings — General

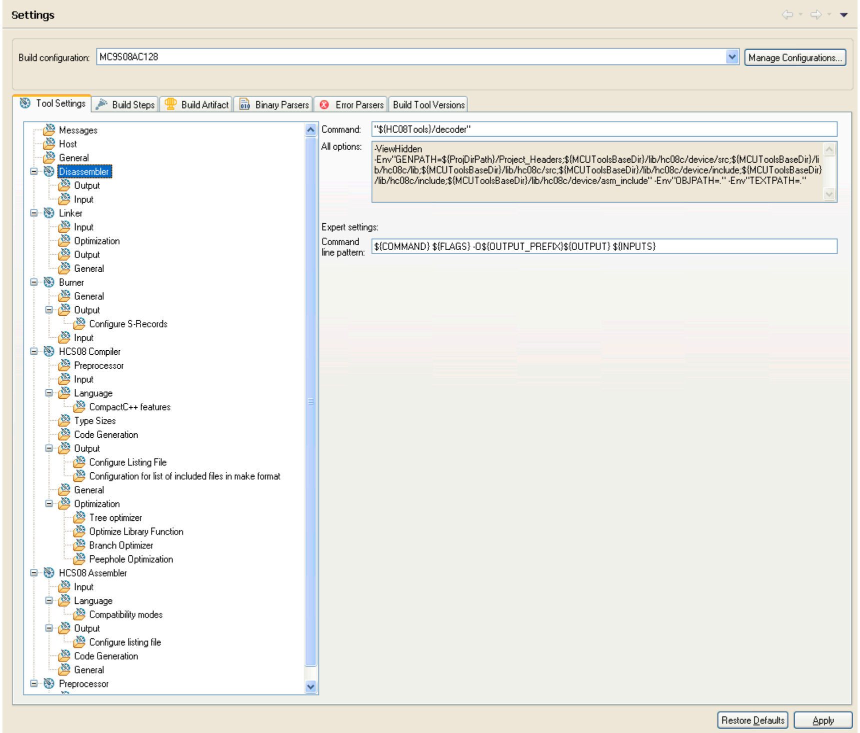
Option	Description
Memory Model (-M)	Specify the memory model for the build tools: • **Tiny** — Assumes that data pointers have 8-bit addresses unless explicitly specified with the keyword __far • **Small** — Default memory model; assumes that all functions and pointers have 16 bit addresses and requires code and data to be located in 64 kilobytes address space • **Banked** — Lets you place program code into atmost 256 pages of 16 kilobytes each, but does not affect data allocation
Enable Memory Management Unit (MMU) Support (-MMU)	Check to inform the compiler that `CALL` and `RTC` instructions are available, enabling code banking, and that the current architecture has extended data access capabilities, enabling support for `__linear` data types. This option can be used only when `-Cs08` is enabled.
Other Flags	Specify additional command line options for the HCS08; type in custom flags that are not otherwise available in the UI.

Disassembler

Use this panel to specify the command, options, and expert settings for HCS08 Disassembler.

Figure 3.9 shows the Disassembler page.

Figure 3.9 Tool Settings > Disassembler


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: "${HC08Tools}/decoder"

All options:

```
-ViewHidden
-Env"GENPATH=${ProjDirPath}/Project_Headers;${MCUToolsBaseDir}/lib/hc08c/device/src;${MCUToolsBaseDir}/li
b/hc08c/lib;${MCUToolsBaseDir}/lib/hc08c/src;${MCUToolsBaseDir}/lib/hc08c/device/include;${MCUToolsBaseDir}
/lib/hc08c/include;${MCUToolsBaseDir}/lib/hc08c/device/asm_include" -Env"OBJPATH=." -Env"TEXTPATH=."
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} -O${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.7 lists and describes the Disassembler options.

Table 3.7 Tool Settings — Disassembler Options

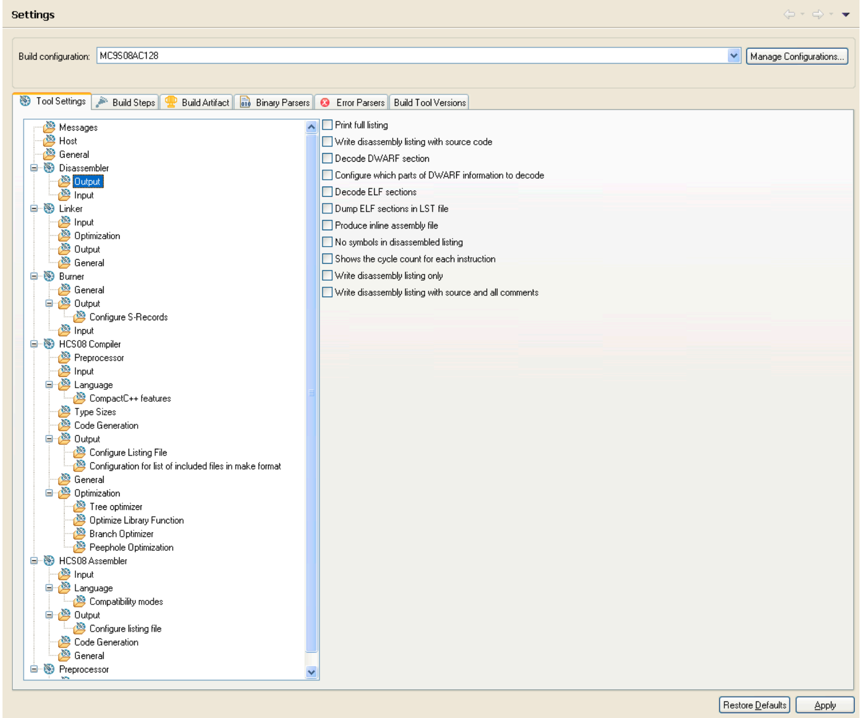
Option	Description
Command	Shows the location of the disassembler executable file; default is `${HC08Tools}/decoder`.
All options	Shows the actual command line the linker will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} -O${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

Disassembler > Output

Use this panel to control how the disassembler generates the output file.

Figure 3.10 shows the **Output** panel.

Figure 3.10 Tool Settings — Disassembler > Output


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Print full listing
☐ Write disassembly listing with source code
☐ Decode DWARF section
☐ Configure which parts of DWARF information to decode
☐ Decode ELF sections
☐ Dump ELF sections in LST file
☐ Produce inline assembly file
☐ No symbols in disassembled listing
☐ Shows the cycle count for each instruction
☐ Write disassembly listing only
☐ Write disassembly listing with source and all comments

Table 3.8 lists and describes the output options for HCS08 disassembler.

Table 3.8 Tool Settings — Disassembler > Output Options

Option	Description
Print full listing	Prints a listing with the header information of the object file.
Write disassembly listing with source code	Check to enable the decoder decoding Freescale object files write the source code within the disassembly listing. This option setting is default for the Freescale object files as input.

Table 3.8 Tool Settings — Disassembler > Output Options (*continued*)

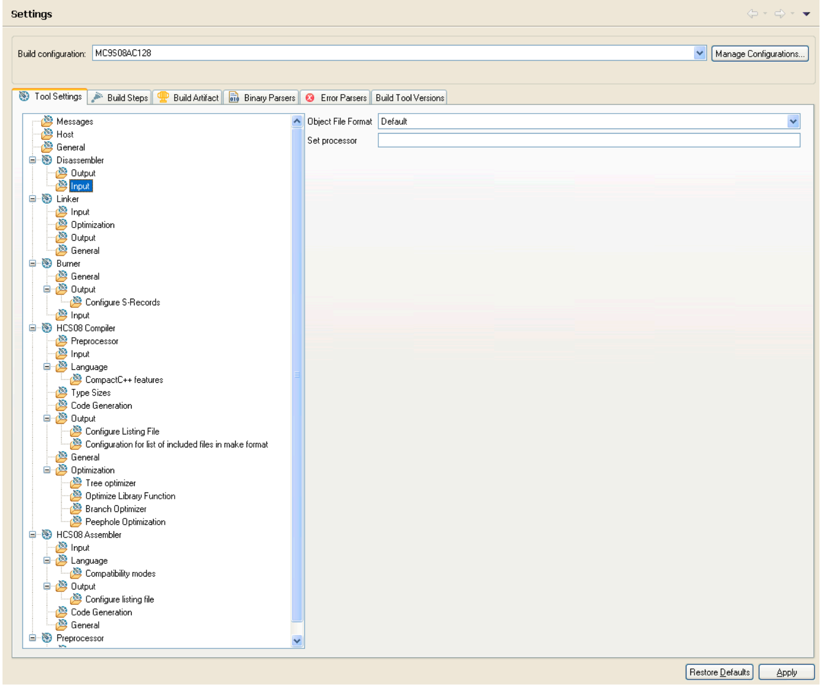
Option	Description
Decode DWARF section	Check to write the DWARF section information in the listing file. Decoding from the DWARF section inserts this information in the listing file.
Configure which parts of DWARF in formation to decode	Check to configure parts of DWARF in formation to decode.
Decode ELF sections	Check to ensure that the ELF section information is also written to the listing file. Decoding from the ELF section inserts the following information in the listing file.
Dump ELF sections	Check to generate a HEX dump of all ELF sections.
Dump ELF sections in LST file	Check to generate a HEX dump of all ELF sections in a LST file.
Produce inline assembly file	Check to ensure that the output listing is an inline assembly file without additional information, but in C comments.
No symbols in disassembled listing	Check to prevent symbols from printing in the disassembled listing.
Shows the cycle count for each instruction	Check to ensure that each instruction line contains the count of cycles in '[',']' braces. The cycle count is written before the mnemonics of the instruction. Note that the cycle count display is not supported for all architectures.
Write disassembly listing only	Check to ensure that the Decoder decoding Freescale object files writes the source code within the disassembly listing only.
Write disassembly listing with source and all comments	Check to write the origin source and its comments within the disassembly listing.

Disassembler > Input

Use this panel to control how the disassembler generates the input file.

Figure 3.11 shows the **Input** panel.

Figure 3.11 Tool Settings — Disassembler > Input


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Object File Format: Default

Set processor:

Table 3.9 lists and describes the input options for HCS08 disassembler.

Table 3.9 Tool Settings — Disassembler > Input Options

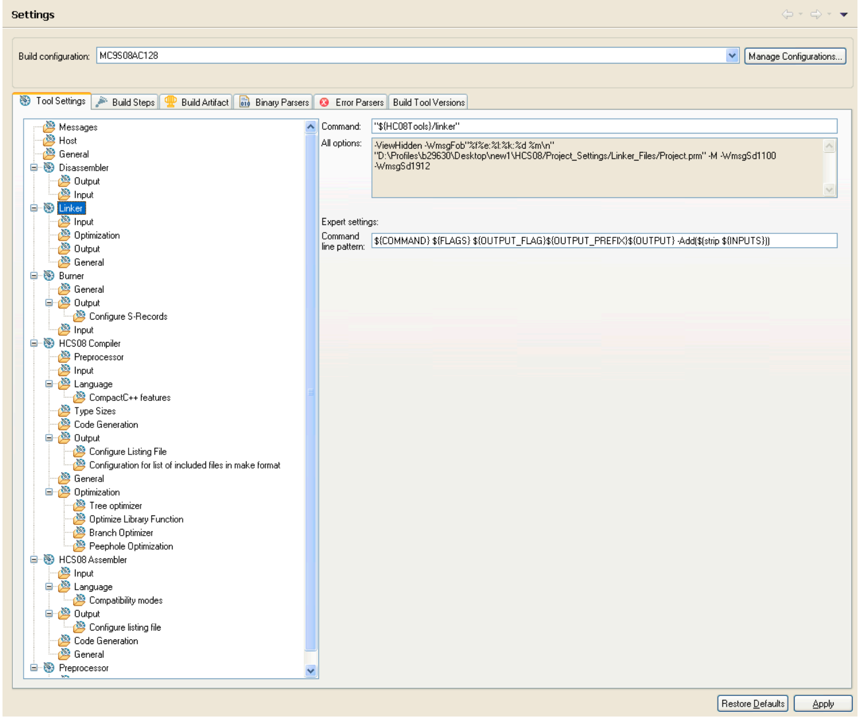
Option	Description
Object File Format	Defines the format for the output file generated by the disassembler.
Set processor	Specifies which processor should be decoded. For object files, libraries and applications, the processor is usually detected automatically. For S-Record and Intel Hex files, however, the decoder cannot determine which CPU the code is for, and therefore the processor must be specified with this option to get a disassembly output. Without this option, only the structure of a S-Record file is decoded. The following values are supported: HC08, HC08:HCS08, HC11, HC12, HC12:CPU12, HC12:HCS12, HC12:HCS12X, HC16, M68k, MCORE, PPC, RS08, 8500, 8300, 8051 and XA.

Linker

Use this panel to specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the general, libraries, and search path settings.

Figure 3.12 shows the **Linker** settings.

Figure 3.12 Tool Settings — Linker


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: `"${HC08Tools}/linker"`

All options:

```
-ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n"
"D:\Profiles\b29630\Desktop\new1\HCS08/Project_Settings/Linker_Files/Project.prm" -M -WmsgSd1100
-WmsgSd1912
```

Expert settings:

Command line pattern: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} -Add($(strip ${INPUTS}))`

[Table 3.10](#) lists and describes the linker options for HCS08.

Table 3.10 Tool Settings — Linker Options

Option	Description
Command	Shows the location of the linker executable file. Default value is `"${HC08Tools}/linker.exe"`
All options	Shows the actual command line the linker will be called with.

Table 3.10 Tool Settings — Linker Options (*continued*)

Option	Description
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} -add( ${INPUTS} )`

Linker > Input

Use this panel to specify the parameter file path, startup function, object file search paths, and any additional libraries that the **C/C++ Linker** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

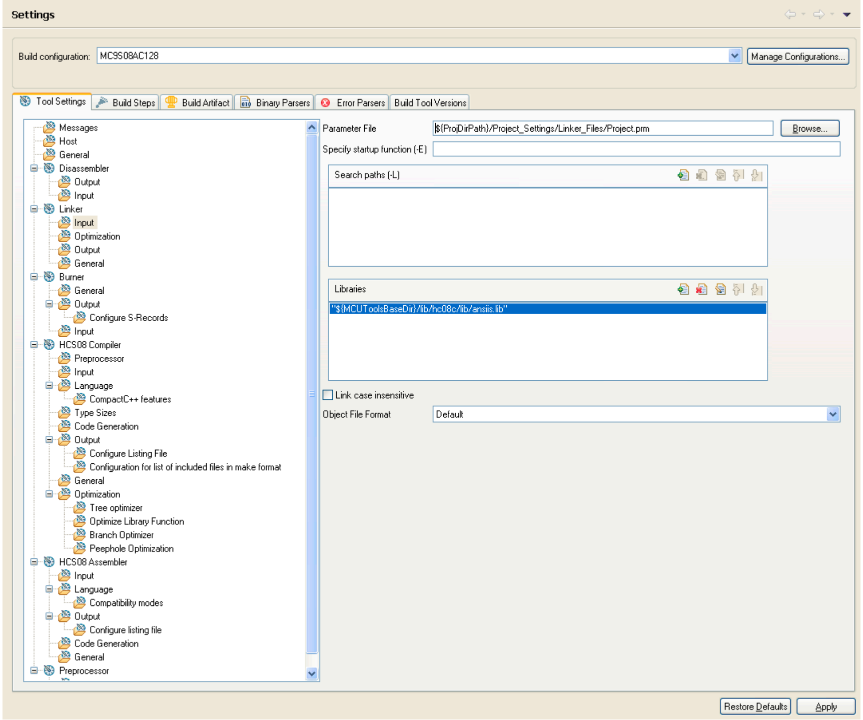
The IDE first looks for an include file in the current directory, or the directory that you specify in the `INCLUDE` directive. If the IDE does not find the file, it continues searching the paths shown in this panel. The IDE keeps searching paths until it finds the `#include` file or finishes searching the last path at the bottom of the **Include File Search Paths** list. The IDE appends to each path the string that you specify in the `INCLUDE` directive.

NOTE The IDE displays an error message if a header file is in a different directory from the referencing source file. Sometimes, the IDE also displays an error message if a header file is in the same directory as the referencing source file.

For example, if you see the message `Could not open source file myfile.h`, you must add the path for `myfile.h` to this panel.

Figure 3.13 shows the **Input** panel.

Figure 3.13 Tool Settings — Linker > Input



Build configuration: MC9S08AC128

Parameter File: ${ProjDirPath}/Project_Settings/Linker_Files/Project.prm

Specify startup function (-E):

Search paths (-L)

Libraries: "${MCUToolsBaseDir}/lib/hc08c/lib/ansiis.lib"

☐ Link case insensitive

Object File Format: Default

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Browse...
Restore Defaults
Apply

Table 3.11 lists and describes the linker input options for HCS08.

Table 3.11 Tool Settings — Linker > Input Options

Option	Description
Parameter File	Shows the path of the parameter file. Default value is `${ProjDirPath}/Project_Settings/Linker_Files/Project.prm`.
Specify startup function (-E)	Specify the path of command-line tool to preprocess source files.
Search paths (-L)	Shows the list of all search paths; the ELF part of the linker searches object files first in all paths and then the usual environment variables are considered.

Table 3.11 Tool Settings — Linker > Input Options (*continued*)

Option	Description
Libraries	Lists paths to additional libraries that the C/C++ linker uses. Default value is `"${MCUToolsBaseDir}/lib/hc08c/lib/ansiis.lib"`
Link case insensitive	With this option, the linker ignores object name capitalization. This option supports case-insensitive linking of assembly modules. Since all identifiers are linked case insensitive, this also affects C or C++ modules. This option only affects the comparison of names of linked objects. Section names or the parsing of the link parameter file are unaffected. They remain case sensitive.
Object File Format	Defines the format for the output file generated by the Linker.

Table 3.12 lists and describes the toolbar buttons that help work with the libraries and the additional object file search paths.

Table 3.12 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.14) and specify the object file search path.
✖	**Delete** — Click to delete the selected object file search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.15) and update the selected object file search path.

Table 3.12 Search Paths Toolbar Buttons (*continued*)

Button	Description
⇧	**Move up** — Click to move the selected object file search path one position higher in the list.
⇩	**Move down** — Click to move the selected object file search path one position lower in the list.

Figure 3.14 shows the **Add directory path** dialog box.

Figure 3.14 Add directory path Dialog Box

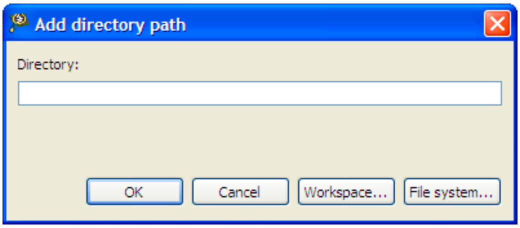
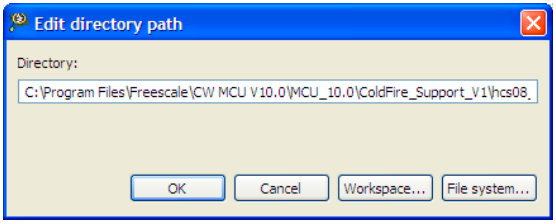


Figure 3.15 shows the **Edit directory path** dialog box.

Figure 3.15 Edit directory path Dialog Box



C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

- **OK —** Click to confirm the action and exit the dialog box.
- **Cancel —** Click to cancel the action and exit the dialog box.
- **Workspace —** Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system —** Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

Table 3.13 lists and describes the toolbar buttons that help work with the libraries and the additional object files.

Table 3.13 Libraries Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add file path** dialog box (Figure 3.16) and specify location of the library you want to add.
✖	**Delete** — Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit file path** dialog box (Figure 3.17) and update the selected path.
⇧	**Move up** — Click to move the selected path one position higher in the list.
⇩	**Move down** — Click to move the selected path one position lower in the list.

Figure 3.16 shows the **Add file path** dialog box.

Figure 3.16 Tool Settings — Linker > Libraries - Add file path Dialog Box

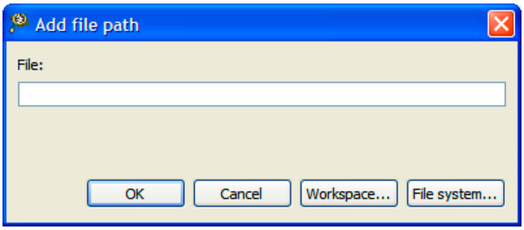
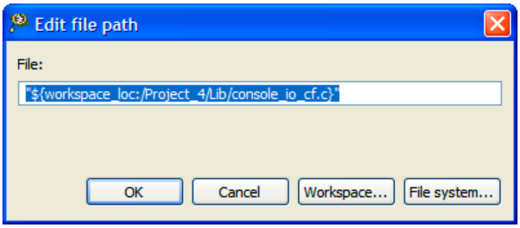


Figure 3.17 shows the **Edit file path** dialog box.

Figure 3.17 Tool Settings — Linker > Libraries - Edit file path Dialog Box



File: `"${workspace_loc:/Project_4/Lib/console_io_cf.c}"`

The buttons in the **Add file path and Edit file path** dialog boxes help work with the file paths.

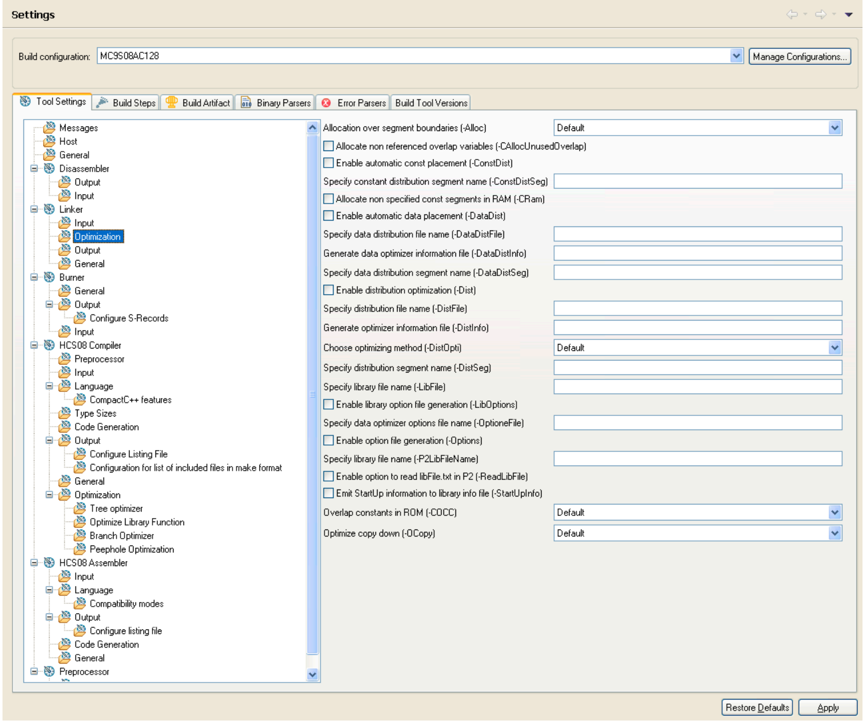
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **File Selection** dialog box and specify the file path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Open** dialog box and specify the file path. The resulting absolute path appears in the appropriate list.

Linker > Optimization

Use this panel to control linker optimizations. The linker's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.18 shows the **Optimization** panel.

Figure 3.18 Tool Settings — Linker > Optimization


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

Allocation over segment boundaries (-Alloc): Default
☐ Allocate non referenced overlap variables (-CAllocUnusedOverlap)
☐ Enable automatic const placement (-ConstDist)
Specify constant distribution segment name (-ConstDistSeg)
☐ Allocate non specified const segments in RAM (-CRam)
☐ Enable automatic data placement (-DataDist)
Specify data distribution file name (-DataDistFile)
Generate data optimizer information file (-DataDistInfo)
Specify data distribution segment name (-DataDistSeg)
☐ Enable distribution optimization (-Dist)
Specify distribution file name (-DistFile)
Generate optimizer information file (-DistInfo)
Choose optimizing method (-DistOpti): Default
Specify distribution segment name (-DistSeg)
Specify library file name (-LibFile)
☐ Enable library option file generation (-LibOptions)
Specify data optimizer options file name (-OptioneFile)
☐ Enable option file generation (-Options)
Specify library file name (-P2LibFileName)
☐ Enable option to read libFile.txt in P2 (-ReadLibFile)
☐ Emit StartUp information to library info file (-StartUpInfo)
Overlap constants in ROM (-COCC): Default
Optimize copy down (-OCopy): Default

Table 3.14 lists and describes the linker optimization options for HCS08.

Table 3.14 Tool Settings — Linker > Optimization Options

Option	Description
Allocation over segment boundaries (-Alloc)	The linker supports to allocate objects from one ELF section into different segments. The allocation strategy controls where space for the next object is allocated as soon as the first segment is full. In the AllocNext strategy, the linker always takes the next segment as soon as the current segment is full. Holes generated during this process are not used later. With this strategy, the allocation order corresponds to the definition order in the object files. Objects defined first in a source file are allocated before later defined objects. In the AllocFirst strategy, the linker checks for every object, if there is a previously only partially used segment, into which the current object does fit. This strategy does not maintain the definition order. In the AllocChange strategy, the linker checks as soon as a object does no longer fit into the current segment, if there is a previously only partially used segment, into which the current object does fit. This strategy does not maintain the definition order, but it does however use fewer different ranges than the AllocFirst case.
Allocate non referenced overlap variables (-CAllocUnusedOverlap)	When Smart Linking is switched off, defined but unreferenced overlapped variables are not allocated by default. Such variables do not belong to a specific function, therefore they cannot be allocated overlapped with other variables. This option only changes the behavior of variables in the special _OVERLAP segment. This segment is used only to allocate parameters and local variables for processors which do not have a stack. Not allocating an unreferenced overlap variable is similar to not allocating a variable on the stack for other processors. If you use this stack analogy, then allocating such variables this way corresponds to allocating unreferenced stack variables in global memory. This option allows allocation of all defined objects. Using this option is not recommended.
Enable automatic const placement (-ConstDist)	With this option the linker constant optimizer is enabled. Instead of performing usual linking actions, the linker generates a data distribution file which contains optimized distribution for constant objects.

Table 3.14 Tool Settings — Linker > Optimization Options (*continued*)

Option	Description
Specify constant distribution segment name (-ConstDistSeg)	When this option is enabled, it's possible to specify the name of the constant distribution segment.
Allcoate non specified const segments in RAM (-CRam)	This option allocates constant data segments not explicitly allocated in a READ_ONLY segment in the default READ_WRITE segment. This was the default for old versions of the linker, so this option provides a compatible behavior with old linker versions.
Enable automatic data placement (-DataDist)	With this option the linker data optimizer is enabled. Instead of performing usual linking actions, the linker generates a data distribution file which contains optimized distribution.
Specify data distribution file name (-DataDistFile)	When this option is enabled, it's possible to specify the name of the data distribution file. There, all distributed data and how the compiler has to reallocate them are listed.
Generate data optimizer information file (-DataDistInfo)	When this option is enabled, the data optimizer generates a data distribution information file giving information on object to segment mapping
Specify data distribution segment name (-DataDistSeg)	When this option is enabled, it's possible to specify the name of the data distribution segment.
Enable distribution optimization (-Dist)	This option enables the linker optimizer. Instead of a link, the linker generates a distribution file which contains an optimized distribution.
Specify distribution file name (-DistFile)	Enable this option to specify the name of the distribution file. The distribution file lists all distributed functions and specifies how the compiler reallocates them.
Generate optimizer information file (-DistInfo)	Using this option, the optimizer generates a distribution information file containing a list of all sections and their functions. Available function information includes the old size, optimized size, and new calling convention.

Table 3.14 Tool Settings — Linker > Optimization Options (*continued*)

Option	Description
Choose optimizing method (-DistOpti)	Enable this option to choose the optimizing method. With the FillBanks argument the linker minimizes the free space in every bank. FillBanks is most effective for functions using the near calling convention. Use the CodeSize argument to minimize code when free space within the banks is no concern.
Specify distribution segment name (-DistSeg)	Use this option to specify the name of the distribution segment.
Specify library file name (-LibFile)	When this option is enabled,linker generates file<filename> which has information about the current libraries and also about the files with which they should be replaced with.
Enable library option file generation (_LibOptions)	Enables library information generation. When this option is enabled,linker generates file (default libFile.txt) which has information about the current library and the startup file and also about the files with which they should be replaced with.
Specify data optimizer options file name (-OptioneFile)	Specifies the name of the file that contains the set of linker-generated compiler options. When this option is enabled, linker places the second step compiler options in the specified file<filename>.
Enable option file generation (-Options)	Enables compiler option generation. The generated options will be used for second step compilation. Linker generates a text file containing a compiler option for the second step (one of the following: -ConstQualiNear, -NonConstQualiNear, -Mb). The content of the file is appended to the compiler options for the second compilation step.
Specify library file name (-P2LibFileName)	Specifies the name of the library information file. When this option is enabled in second link step,linker reads file<filename> which has information about the libraries.
Enable option to read libFile.txt in P2 (-ReadLibFile)	Instructs the linker to read in the library information file that it generated in step one. This option is passed in second link step. It tells the linker to read library information file(default libFile.txt).

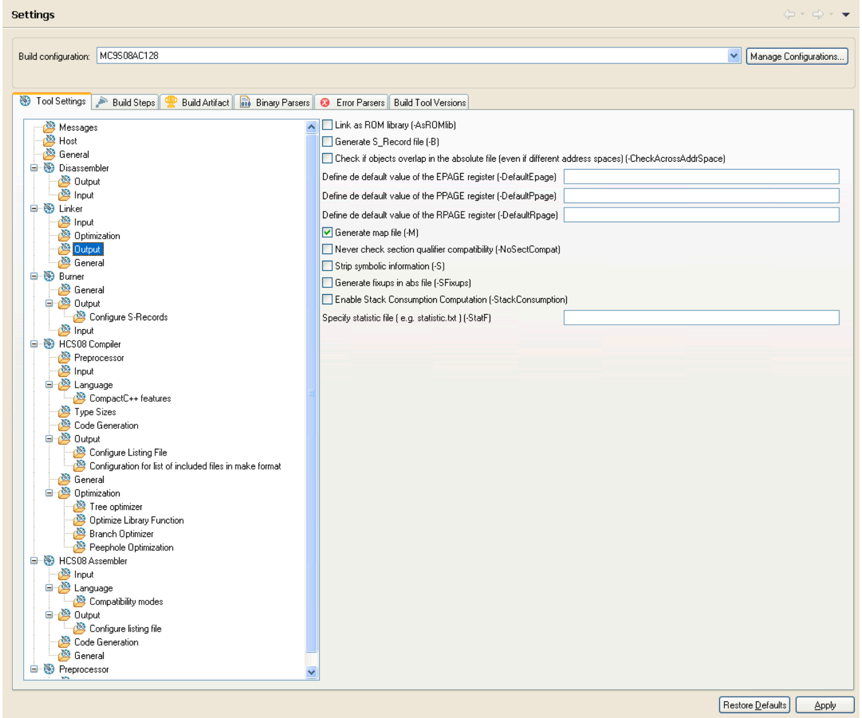
Table 3.14 Tool Settings — Linker > Optimization Options (*continued*)

Option	Description
Emit StartUp information to library info file (-StartUpInfo)	The information about the current startup file and the replacement startup file will be added to the library file(default libFile.txt) and used during the second compile-link step.
Overlap constants in ROM (-COCC)	Defines the default if constants and code should be optimized; commands `DO_OVERLAP_CONSTS` and `DO_NOT_OVERLAP_CONSTS` take precedence over the option.
Optimize copy down (-OCopy)	Changes the copy down structure to use few spaces. The optimization does assume that the application does perform both the zero out and the copy down step of the global initialization. If a value is set to zero by the zero out, then zero values are removed from the copy down information. The resulting initialization is not changed by this optimization if the default startup code is used.

Linker > Output

Use this panel to control how the linker formats the listing file, as well as error and warning messages. Figure 3.19 shows the **Output** panel.

Figure 3.19 Tool Settings — Linker > Output


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Link as ROM library (-AsROMlib)
☐ Generate S_Record file (-B)
☐ Check if objects overlap in the absolute file (even if different address spaces) (-CheckAcrossAddrSpace)
Define de default value of the EPAGE register (-DefaultEpage)
Define de default value of the PPAGE register (-DefaultPpage)
Define de default value of the RPAGE register (-DefaultRpage)
☑ Generate map file (-M)
☐ Never check section qualifier compatibility (-NoSectCompat)
☐ Strip symbolic information (-S)
☐ Generate fixups in abs file (-SFixups)
☐ Enable Stack Consumption Computation (-StackConsumption)
Specify statistic file (e.g. statistic.txt) (-StatF)

Table 3.15 lists and describes the linker output options for HCS08.

Table 3.15 Tool Settings — Linker > Output Options

Option	Description
Link as ROM library (-AsROMlib)	Check to link the application as a ROM library. This option has the same effect as specifying AS ROM_LIB in the linker parameter file.
Generate S_record file (-B)	Check to specify that in addition to an absolute file, also an srecord file should be generated. The name of the srecord file is the same as the name of the abs file, except that the extension "SX" is used. The default.env variable "SRECORD" may specify an alternative extension.

Table 3.15 Tool Settings — Linker > Output Options (*continued*)

Option	Description
Check if objects overlap in the absolute file (even if different address spaces) (-CheckAcrossAddrSpace)	Check to instruct the linker to check if objects overlap, taking into account their address space.
Define de default value of the EPAGE register (-DefaultEpage)	Defines the reset value for the EEPROM Page Index Register (EPAGE). The value is specific to the actual S12(X) derivative.
Define de default value of the PPAGE register (-DefaultPpage)	Defines the reset value for the Program Page Index Register (PPAGE). The value is specific to the actual S12(X) derivative.
Define de default value of the RPAGE register (-DefaultRpage)	Defines the reset value for the RAM Page Index Register (RPAGE). The value is specific to the actual S12(X) derivative.
Generate map file (-M)	Check to scan source files for dependencies and emit a Makefile, without generating object code.
Never check section qualifier compatibility (-NoSectCompat)	For some target CPU's, when placing a section in a segment the linker checks if the qualifiers of the section are compatible with the ones of the segment (for instance when placing .text into RAM may result in a linker error).This option disables the check.
Strip symbolic information (-S)	Check to disable the generation of DWARF sections in the absolute file to save memory space.
Generate fixups in abs file (-SFixups)	Check to ensure compatibility with previous linker versions. Usually, absolute files do not contain any fixups because all fixups are evaluated at link time. But with fixups, the decoder might symbolically decode the content in absolute files. Some debuggers do not load absolute files which contain fixups because they assume that these fixups are not yet evaluated. But the fixups inserted with this option are actually already handled by this linker.

Table 3.15 Tool Settings — Linker > Output Options (*continued*)

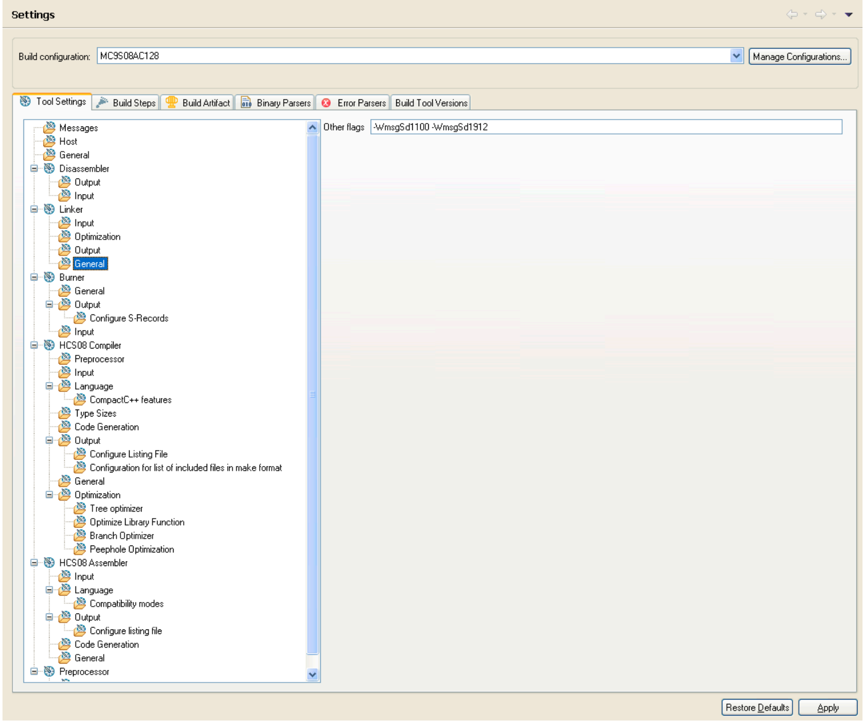
Option	Description
Enable Stack Consumption Computation (-StackConsumption)	The linker computes maximum stack effect for given application when the option is enabled and places the result in the output .map file.
Specify statistic file (e.g. statistic.txt) (-StatF)	Specify the name of the linker statistic file. The statistic file reports each allocated object and its attributes. Every attribute is separated by a tab character, so it can be easily imported into a spreadsheet/database program for further processing.

Linker > General

Use this panel to specify the general linker behavior.

Figure 3.20 shows the **General** panel.

Figure 3.20 Tool Settings — Linker > General


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

Other flags: -WmsgSd1100 -WmsgSd1912

Table 3.16 lists and describes the **general linker options for HCS08.**

Table 3.16 Tool Settings — Linker > General Options

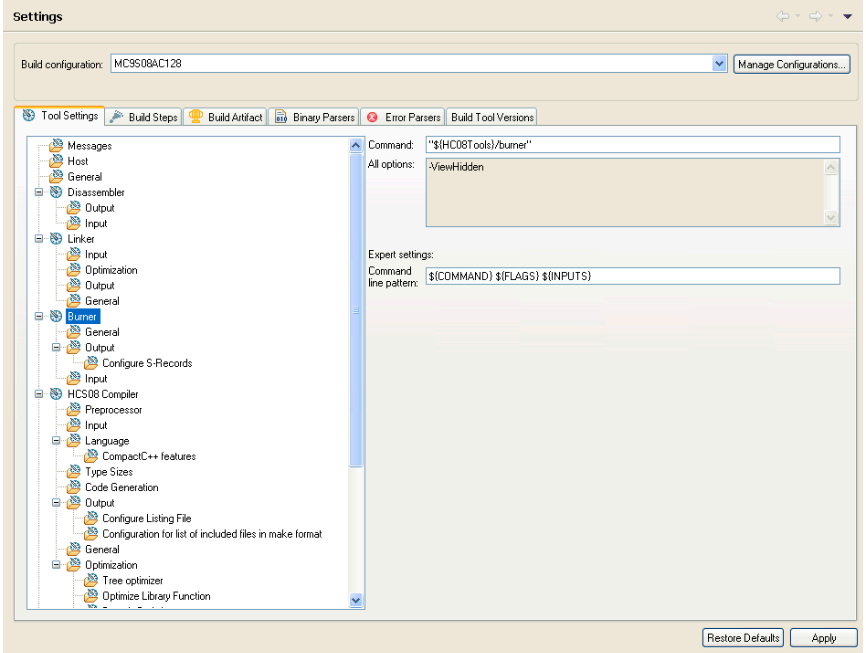
Option	Description
Other flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI. Default value is: `-WmsgSd1100 -WmsgSd1912`

Burner

Use the Burner for HCS08 Preference Panel to map *.bbl (batch burner language) files to the Burner Plug-In. When the project folder contains a *.bbl file, *.bbl file processing during the post-link phase uses the settings in the Burner preference panel.

Figure 3.21 shows the HCS08 Burner settings.

Figure 3.21 Tool Settings > Burner


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: "${HC08Tools}/burner"

All options: -ViewHidden

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${INPUTS}

Table 3.17 lists and describes the burner options for HCS08.

Table 3.17 Tool Settings — Burner Options

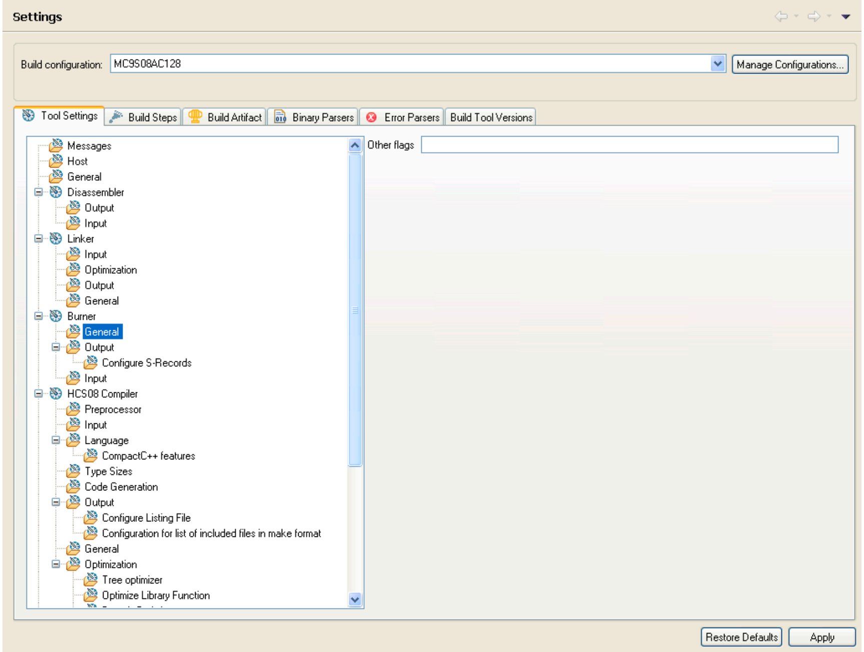
Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${HC08Tools}/burner"`
All options	Shows the actual command line the burner will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${INPUTS}`.

Burner > General

Use this panel to specify other flags for the HCS08 Burner to use.

Figure 3.22 shows the **General** panel.

Figure 3.22 Tool Settings — Burner > General



Build configuration: MC9S08AC128

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Other flags
Restore Defaults
Apply

Table 3.18 lists and describes the general options for HCS08 burner.

Table 3.18 Tool Settings — Burner > General Options

Option	Description
Other flags	Specify additional command line options for the burner; type in custom flags that are not otherwise available in the UI.

Burner > Output

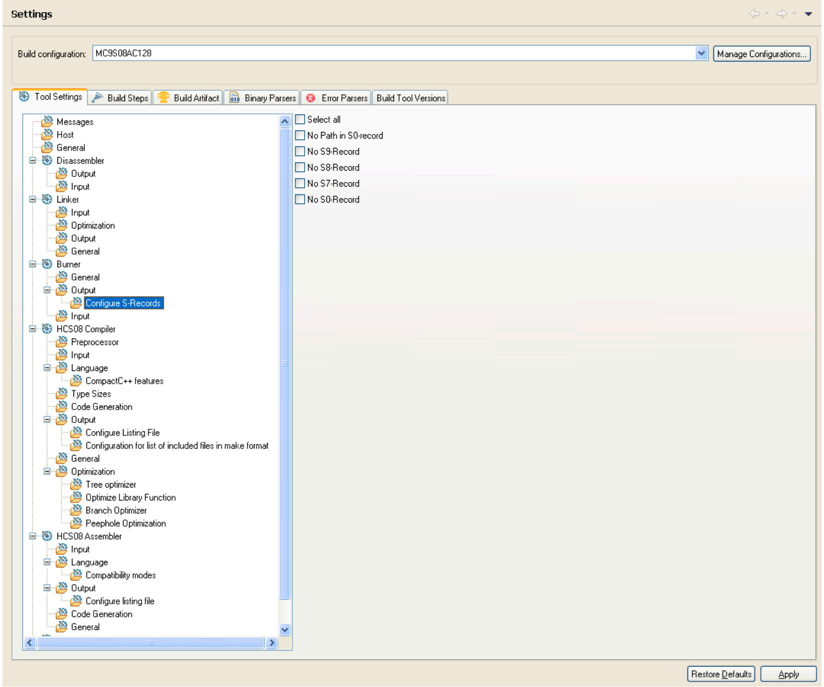
Use this panel to specify the output options for HCS08.

Burner > Output > Configure S-Record

Use this panel to configure the S-record options of the Burner.

Figure 3.23 shows the **Configure S-Record** panel.

Figure 3.23 Tool Settings — Burner > Configure S-Record



Build configuration: MC9S08AC128

☐ Select all
☐ No Path in S0-record
☐ No S9-Record
☐ No S8-Record
☐ No S7-Record
☐ No S0-Record

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Restore Defaults
Apply

Table 3.19 lists and describes the general options for HCS08 configure S-Record.

Table 3.19 Tool Settings — Burner > Output > Configure S-Record Options

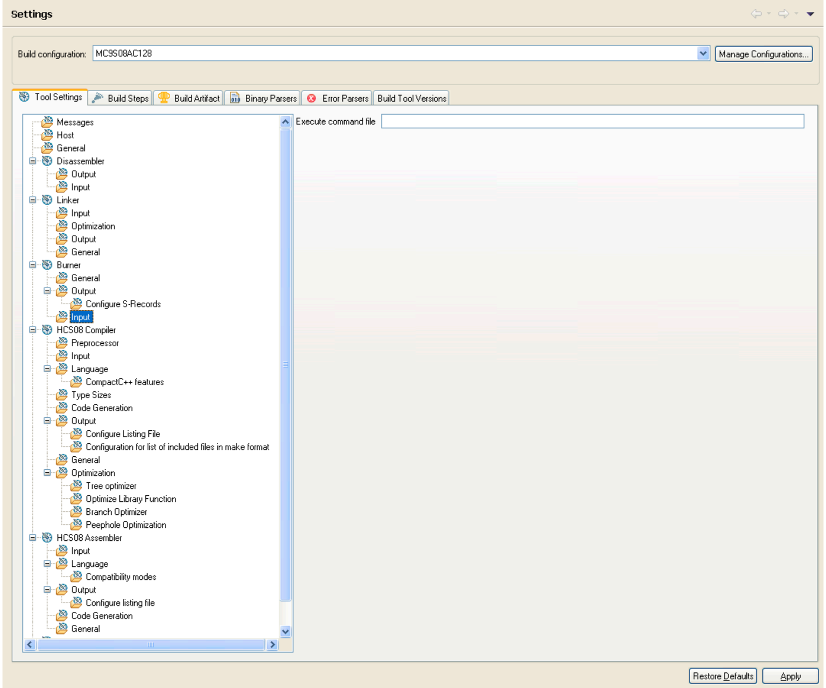
Option	Description
Select all	Disables generation of all start (S0) and end records (S7, S8, or S9)
No path in S0-record	Removes the path (if present) from the file name in the S0 record
No S9-record	Disables generation of S9-record
No S8-record	Disables generation of S8-record
No S7-record	Disables generation of S7-record
No S0-record	Disables generation of S0-record

Burner > Input

Use this panel to specify the execute command file of the Burner input.

Figure 3.24 shows the **Input** panel.

Figure 3.24 Tool Settings — Burner > Input


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Execute command file
Restore Defaults
Apply

Build configuration: MC9S08AC128

Table 3.20 lists and describes the inputs options for burner.

Table 3.20 Tool Settings — Burner > Input Options

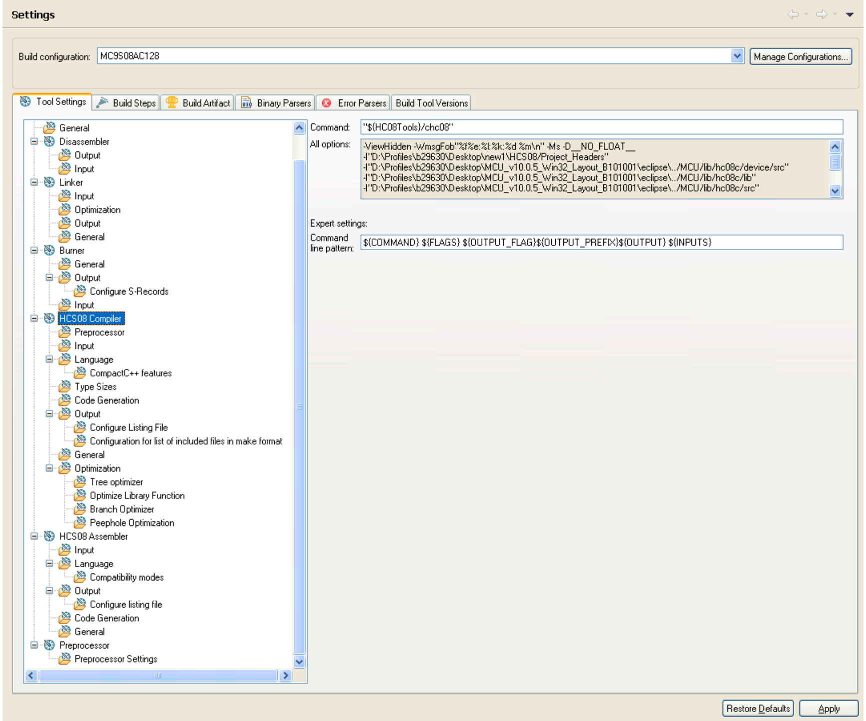
Option	Description
Execute command file	This option causes the Burner to execute a Batch Burner command file (usual extension is .bbl).

HCS08 Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the HCS08 Compiler tree control includes the general and the file search path settings.

Figure 3.25 shows the HCS08 Compiler settings.

Figure 3.25 Tool Settings — HCS08 Compiler



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: "${HC08Tools}/chc08"

All options:

```
-ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -Ms -D__NO_FLOAT__
-I"D:\Profiles\b29630\Desktop\new1\HCS08/Project_Headers"
-I"D:\Profiles\b29630\Desktop\MCU_v10.0.5_Win32_Layout_B101001\eclipse\../MCU/lib/hc08c/device/src"
-I"D:\Profiles\b29630\Desktop\MCU_v10.0.5_Win32_Layout_B101001\eclipse\../MCU/lib/hc08c/lib"
-I"D:\Profiles\b29630\Desktop\MCU_v10.0.5_Win32_Layout_B101001\eclipse\../MCU/lib/hc08c/src"
```

Expert settings:

Command line pattern: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

Table 3.21 lists and describes the compiler options for HCS08.

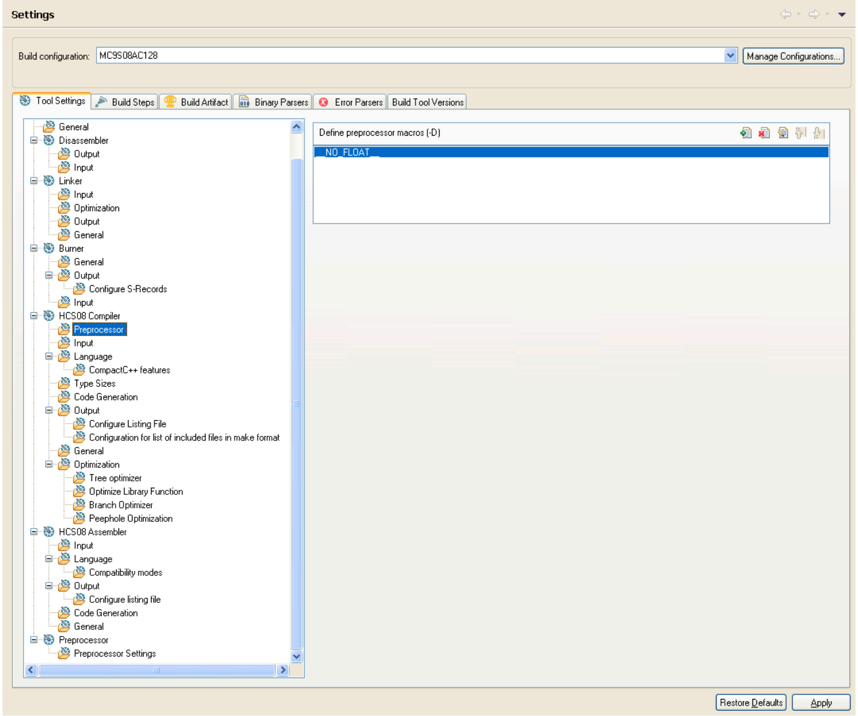
Table 3.21 Tool Settings — Compiler Options

Option	Description
Command	Shows the location of the linker executable file. Default value is:`"${HC08Tools}/chc08.exe"`
All options	Shows the actual command line the compiler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS}${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

HCS08 Compiler > Preprocessor

Use this panel to specify preprocessor behavior. You can specify the file paths and define macros. Figure 3.26 shows the **Preprocessor** panel.

Figure 3.26 Tool Settings — HCS08 Compiler > Preprocessor


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

Define preprocessor macros (-D)

__NO_FLOAT__

Table 3.22 lists and describes the preprocessor options for HCS08 Compiler.

Table 3.22 Tool Settings — HCS08 Compiler > Preprocessor Option

Option	Description
Define preprocessor macros (-D)	Define, delete, or rearrange preprocessor macros. You can specify multiple macros and change the order in which the IDE uses the macros. Define preprocessor macros and optionally assign their values. This setting is equivalent to specifying the `-D name[=value]` command-line option. To assign a value, use the equal sign (=) with no white space. For example, this syntax defines a preprocessor value named `EXTENDED_FEATURE` and assigns `ON` as its value: `EXTENDED_FEATURE=ON` Note: If you do not assign a value to the macro, the shell assigns a default value of 1.

Table 3.23 lists and describes the toolbar buttons that help work with preprocessor macro definitions.

Table 3.23 Define Preprocessor Macros Toolbar Buttons

Button	Description
✚	**Add —** Click to open the **Enter Value** dialog box (Figure 3.27) and specify the path/macro.
✖	**Delete —** Click to delete the selected path/ macro. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit Dialog** dialog box (Figure 3.28) and update the selected path/macro.

Table 3.23 Define Preprocessor Macros Toolbar Buttons (*continued*)

Button	Description
⇧	**Move up** — Click to move the selected path/macro one position higher in the list.
⇩	**Move down** — Click to move the selected path/macro one position lower in the list.

Figure 3.27 shows the **Enter Value** dialog box.

Figure 3.27 Tool Settings — HCS08 Compiler > Preprocessor - Enter Value Dialog Box

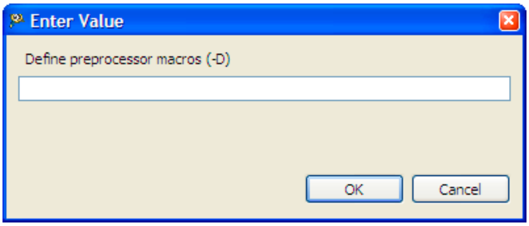
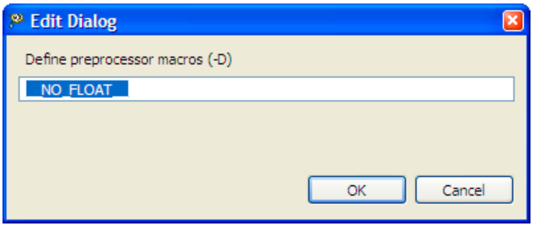


Figure 3.28 shows the **Edit Dialog** dialog box.

Figure 3.28 Tool Settings — HCS08 Compiler > Preprocessor - Edit Dialog Box



Define preprocessor macros (-D): NO_FLOAT

The buttons in the **Enter Value and Edit** dialog boxes help work with the preprocessor macros.

- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.

HCS08 Compiler > Input

Use this panel to specify file search paths and any additional include files the **HCS08 Compiler** should use. You can specify multiple search paths and the order in which you want to perform the search.

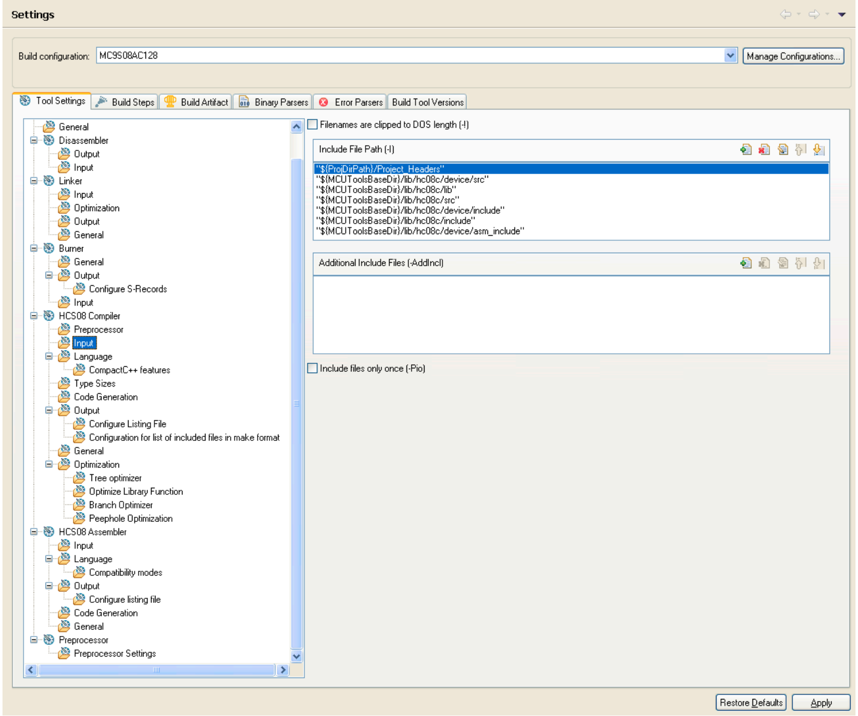
The IDE first looks for an include file in the current directory, or the directory that you specify in the `INCLUDE` directive. If the IDE does not find the file, it continues searching the paths shown in this panel. The IDE keeps searching paths until it finds the `#include` file or finishes searching the last path at the bottom of the Include File Search Paths list. The IDE appends to each path the string that you specify in the `INCLUDE` directive.

NOTE The IDE displays an error message if a header file is in a different directory from the referencing source file. Sometimes, the IDE also displays an error message if a header file is in the same directory as the referencing source file.

For example, if you see the message `Could not open source file myfile.h`, you must add the path for `myfile.h` to this panel.

Figure 3.29 shows the **Input** panel.

Figure 3.29 Tool Settings — HCS08 Compiler > Input


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Filenames are clipped to DOS length (-!)

Include File Path (-I)

```
"${ProjDirPath}/Project_Headers"
"${MCUToolsBaseDir}/lib/hc08c/device/src"
"${MCUToolsBaseDir}/lib/hc08c/lib"
"${MCUToolsBaseDir}/lib/hc08c/src"
"${MCUToolsBaseDir}/lib/hc08c/device/include"
"${MCUToolsBaseDir}/lib/hc08c/include"
"${MCUToolsBaseDir}/lib/hc08c/device/asm_include"
```

Additional Include Files (-AddIncl)

☐ Include files only once (-Pio)

Table 3.24 lists and describes the input options for HCS08 Compiler.

Table 3.24 Tool Settings — HCS08 Compiler > Input Options

Option	Description
Include File Path (-I)	Specify, delete, or rearrange file search paths.
Additional Include Files (-AddInd)	Specify, delete, or rearrange paths to search any additional #include files.
Include files only once (-Pio)	Check to include every header file only once; duplicates are ignored.

Table 3.25 lists and describes the toolbar buttons that help work with the file paths.

Table 3.25 Include File Path (-I) Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.14) and specify location of the library you want to add.
✖	**Delete** — Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.15) and update the selected path.
⇧	**Move up** — Click to move the selected path one position higher in the list.
⇩	**Move down** — Click to move the selected path one position lower in the list.

Table 3.26 lists and describes the toolbar buttons that help work with the search paths.

Table 3.26 Additional Include Files (-AddIncl) Toolbar Buttons

Button	Description
+	**Add —** Click to open the **Add directory path** dialog box (Figure 3.30) and specify location of the library you want to add.
✗	**Delete —** Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.31) and update the selected path.
↑	**Move up —** Click to move the selected path one position higher in the list.
↓	**Move down —** Click to move the selected path one position lower in the list.

Figure 3.30 Tool Settings — HCS08 Compiler > Input - Add file path Dialog Box

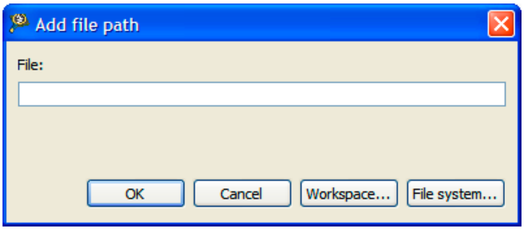
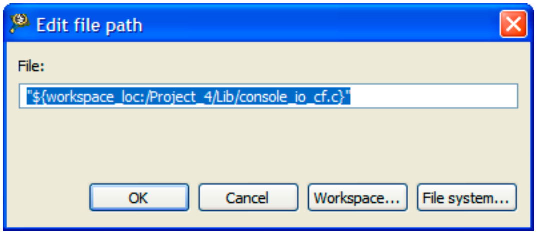


Figure 3.31 Tool Settings — HCS08 Compiler > Input - Edit file path Dialog Box



File: "${workspace_loc:/Project_4/Lib/console_io_cf.c}"

The buttons in the **Add file path** (Figure 3.30) **and Edit file path** (Figure 3.31) dialog boxes help work with the paths.

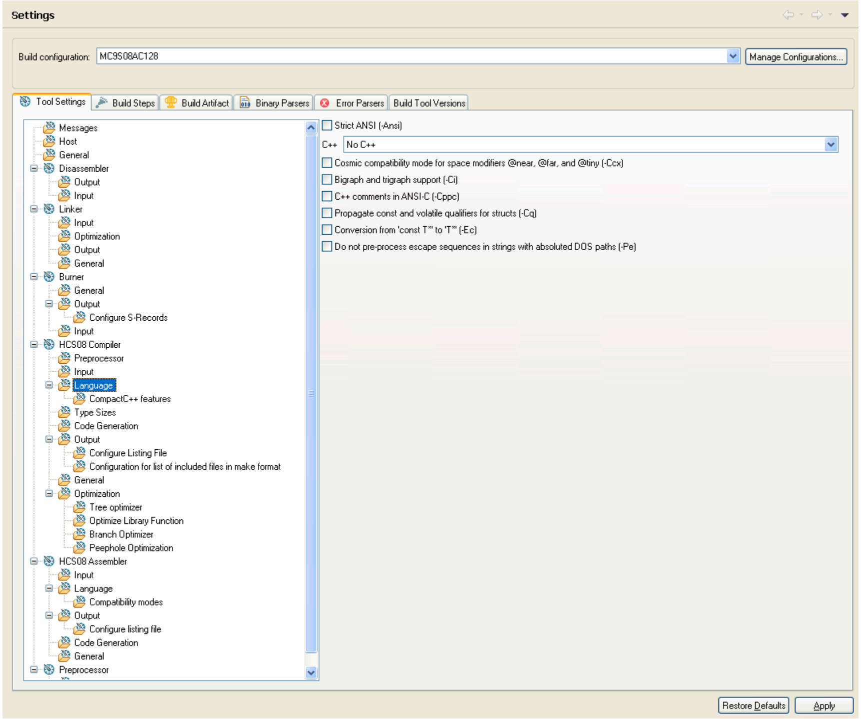
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **File Selection** dialog box and specify the path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Open** dialog box and specify the path. The resulting path appears in the appropriate list.

HCS08 Compiler > Language

Use this panel to specify code- and symbol-generation options for the HCS08 Compiler.

Figure 3.32 shows the **Language** panel.

Figure 3.32 Tool Settings — HCS08 Compiler > Language



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Strict ANSI (-Ansi)

C++ No C++

☐ Cosmic compatibility mode for space modifiers @near, @far, and @tiny (-Ccx)

☐ Bigraph and trigraph support (-Ci)

☐ C++ comments in ANSI-C (-Cppc)

☐ Propagate const and volatile qualifiers for structs (-Cq)

☐ Conversion from 'const T*' to 'T*' (-Ec)

☐ Do not pre-process escape sequences in strings with absoluted DOS paths (-Pe)

Table 3.27 lists and describes the language options for HCS08.

Table 3.27 Tool Settings — HCS08 Compiler > Language Options

Option	Description
Strict ANSI (-Ansi)	Check if you want the C compiler to operate in strict ANSI mode. In this mode, the compiler strictly applies the rules of the ANSI/ISO specification to all input files. This setting is equivalent to specifying the `-ansi` command-line option. The compiler issues a warning for each ANSI/ISO extension it finds.
C++	With this option enabled, the Compiler behaves as a C++ Compiler. You can select between three different types of C++: • **Full C++ (-C++f) —** Supports the whole C++ language. • **Embedded C++ (-C++e)** — Supports a constant subset of the C++ language. EC++ does not support inefficient things like templates, multiple inheritance, virtual base classes and exception handling. • **CompactC++ (-C++c) —** Supports a configurable subset of the C++ language. You can configure this subset with the option -Cn. • **No C++** — If the option is not set, the Compiler behaves as an ANSI-C Compiler. If the option is enabled and the source file name extension is *.c, the Compiler behaves as a C++ Compiler. If the option is not set, but the source filename extension is .cpp or .cxx, the Compiler behaves as if the -C++f option were set.

Table 3.27 Tool Settings — HCS08 Compiler > Language Options (*continued*)

Option	Description
Cosmic compatibility mode for space modifiers @near, @far, and @tiny (-Ccx)	Check to allow Cosmic style @near, @far and @tiny space modifiers as well as @interrupt in your C code. The -ANSI option must be switched off. It is not necessary to remove the Cosmic space modifiers from your application code. There is no need to place the objects to sections addressable by the Cosmic space modifiers. The following is done when a Cosmic modifier is parsed: The objects declared with the space modifier are always allocated in a special Cosmic compatibility (_CX) section (regardless of which section pragma is set) depending on the space modifier, on the const qualifier or if it is a function or a variable. Space modifiers on the left hand side of a pointer declaration specify the pointer type and pointer size, depending on the target.
Bigraph and trigraph support (-Ci)	Check to replace certain unavailable tokens with the equivalent keywords.
C++ comments in ANSI-C (-Cppc)	Check to allow C++ comments.
Propagate const and colatile qualifiers for structs (-Cq)	Check to propagate const and volatile qualifiers for structures. If all members of a structure are constant or volatile, the structure itself is constant or volatile. If the structure is declared as constant or volatile, all its members are constant or volatile, respectively.

Table 3.27 Tool Settings — HCS08 Compiler > Language Options (*continued*)

Option	Description
Conversion from 'const T*' to 'T*' (-Ec)	Check to enable this non-ANSI compliant extension allows the compiler to treat a pointer to a constant type like a pointer to the non-constant equivalent of the type. Earlier Compilers did not check a store to a constant object through a pointer. This option is useful when compiling older source code.
Do not pre-process escape sequences in strings with absolute DOS path (-Pe)	When the -Pe option is enabled, the Compiler handles strings in include directives differently from other strings. Escape sequences in include directive strings are not evaluated. The following example: `#include "C:\names.h"` results in exactly the same include filename as in the source file ("`C:\names.h`"). If the filename appears in a macro, the Compiler does not distinguish between filename usage and normal string usage with escape sequence.

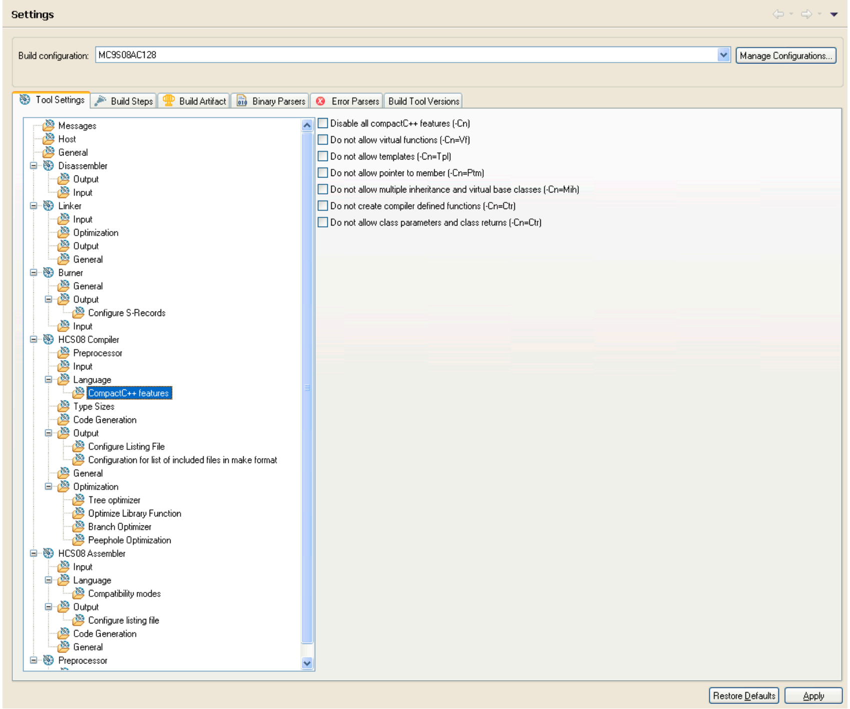
NOTE For more information about the `-Pe` option, refer Microcontrollers V10.x HC08 Build Tools Reference Manual.

HCS08 Compiler > Language > CompactC++ features

Use this panel to select compact C++ features of HCS08 compiler.

Figure 3.33 shows the **Compact C++ features** panel.

Figure 3.33 Tool Settings — HCS08 Compiler > Language > CompactC++ features


Settings
Build configuration: MC9S08AC128
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

☐ Disable all compactC++ features (-Cn)
☐ Do not allow virtual functions (-Cn=Vf)
☐ Do not allow templates (-Cn=Tpl)
☐ Do not allow pointer to member (-Cn=Ptm)
☐ Do not allow multiple inheritance and virtual base classes (-Cn=Mih)
☐ Do not create compiler defined functions (-Cn=Ctr)
☐ Do not allow class parameters and class returns (-Cn=Ctr)

Table 3.28 lists and describes the compactC++ options for HCS08.

Table 3.28 Tool Settings — HCS08 Compiler > Language > CompactC++ Features Options

Option	Description
Disable all compactC++ features (-Cn)	If the -C++ option is enabled, you can disable the compactC++ features. • Vf : Virtual functions are not allowed. Avoid having virtual tables that consume a lot of memory. • Tpl : Templates are not allowed. Avoid having many generated functions perform similar operations. • Ptm : Pointer to member not allowed. Avoid having pointer-to-member objects that consume a lot of memory. • Mih : Multiple inheritance is not allowed. Avoid having complex class hierarchies. Because virtual base classes are logical only when used with multiple inheritance, they are also not allowed. • Ctr : The C++ Compiler can generate several kinds of functions, if necessary: – Default Constructor – Copy Constructor – Destructor – Assignment operator With this option enabled, the Compiler does not create those functions. This is useful when compiling C sources with the C++ Compiler, assuming you do not want C structures to acquire member functions. • Cpr : Class parameters and class returns are not allowed. Avoid overhead with Copy Constructor and Destructor calls when passing parameters, and passing return values of class type.
Do not allow virtual functions (-Cn=Vf)	Virtual functions are not allowed. Avoid having virtual tables that consume a lot of memory
Do not allow templates (-Cn=Tpl)	Templates are not allowed. Avoid having many generated functions perform similar operations.

Table 3.28 Tool Settings — HCS08 Compiler > Language > CompactC++ Features Options (*continued*)

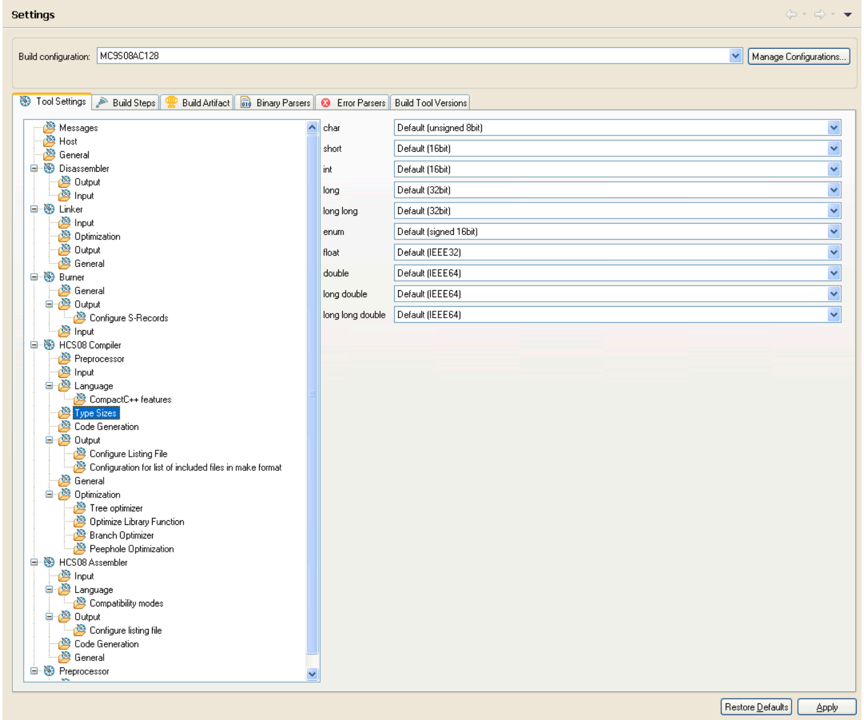
Option	Description
Do not allow pointer to member (-Cn=Ptm)	Pointer to member not allowed. Avoid having pointer-to-member objects that consume a lot of memory.
Do not allow multiple inheritance and virtual base classes (-Cn=Mih)	Multiple inheritance is not allowed. Avoid having complex class hierarchies. Because virtual base classes are logical only when used with multiple inheritance, they are also not allowed.
Do not create compiler defined functions (-Cn=Ctr)	The C++ Compiler can generate several kinds of functions, if necessary: • Default Constructor • Copy Constructor • Destructor • Assignment operator With this option enabled, the Compiler does not create those functions. This is useful when compiling C sources with the C++ Compiler, assuming you do not want C structures to acquire member functions.
Do not allow class parameters and class returns (-Cn=Ctr)	Class parameters and class returns are not allowed. Avoid overhead with Copy Constructor and Destructor calls when passing parameters, and passing return values of class type.

HCS08 Compiler > Type Sizes

Use this panel to specify the available data type size options for the HCS08 Compiler.

Figure 3.34 shows the **Type Sizes** panel.

Figure 3.34 Tool Settings — HCS08 Compiler > Type Sizes


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

char: Default (unsigned 8bit)
short: Default (16bit)
int: Default (16bit)
long: Default (32bit)
long long: Default (32bit)
enum: Default (signed 16bit)
float: Default (IEEE32)
double: Default (IEEE64)
long double: Default (IEEE64)
long long double: Default (IEEE64)

Table 3.29 lists and describes the possible type size options for HCS08 Compiler using the `-T` option.

Table 3.29 Tool Settings — HCS08 Compiler > Type Sizes

Option	Description
char	Selects the size of the `char` type. Options are: • Default (unsigned 8bit) • unsigned 8bit (`-TuCC1`) • signed 8bit (`-TsCC1`) • signed 16bit (`-TsCC2`) • signed 32bit (`-TsCC4`)
short	Selects the size of the `short` type. Options are: • Default (16bit) • signed 8bit (`-TS1`) • signed 16bit (`-TS2`) • signed 32bit (`-TS4`)
int	Selects the size of the `int` type. Options are: • Default (16bit) • signed 8bit (`-TI1`) • signed 16bit (`-TI2`) • signed 32bit (`-TI4`)
long	Selects the size of the `long` type. Options are: • Default (32bit) • signed 8bit (`-TL1`) • signed 16bit (`-TL2`) • signed 32bit (`-TL4`)
long long	Selects the size of the `long long` type. Options are: • Default (32bit) • signed 8bit (`-TLL1`) • signed 16bit (`-TLL2`) • signed 32bit (`-TLL4`)

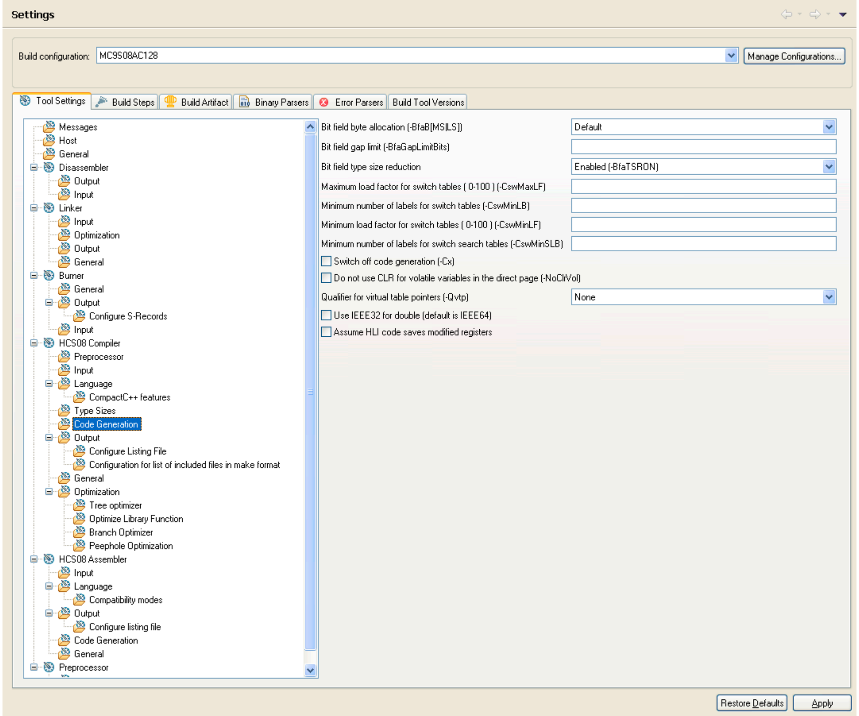
Table 3.29 Tool Settings — HCS08 Compiler > Type Sizes (*continued*)

Option	Description
enum	Selects the size of the `enum` type. Options are: • Default (signed 16bit) • signed 8bit (`-TE1sE`) • signed 16bit (`-TE2sE`) • signed 32bit (`-TE4sE`) • unsigned 8bit (`-TE1uE`)
float	Selects the size of the `float` type. Options are: • Default (IEEE32) • IEEE32 • IEEE64
double	Selects the size of the `double` type. Options are: • Default (IEEE64) • IEEE32 • IEEE64
long double	Selects the size of the `long double` type. Options are: • Default (IEEE64) • IEEE32 • IEEE64
long long double	Selects the size of the `long long double` type. Options are: • Default (IEEE64) • IEEE32 • IEEE64

HCS08 Compiler > Code Generation

Use this panel to specify code- and symbol-generation options for the HCS08 Compiler Figure 3.35 shows the **Code Generation** panel.

Figure 3.35 Tool Settings — HCS08 Compiler > Code Generation


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Build configuration: MC9S08AC128

Bit field byte allocation (-BfaB[MS|LS]) Default

Bit field gap limit (-BfaGapLimitBits)

Bit field type size reduction Enabled (-BfaTSRON)

Maximum load factor for switch tables (0-100) (-CswMaxLF)

Minimum number of labels for switch tables (-CswMinLB)

Minimum load factor for switch tables (0-100) (-CswMinLF)

Minimum number of labels for switch search tables (-CswMinSLB)

☐ Switch off code generation (-Cx)

☐ Do not use CLR for volatile variables in the direct page (-NoClrVol)

Qualifier for virtual table pointers (-Qvtp) None

☐ Use IEEE32 for double (default is IEEE64)

☐ Assume HLI code saves modified registers

Table 3.30 lists and describes the code generation options for HCS08 compiler.

Table 3.30 Tool Settings — HCS08 Compiler > Code Generation Options

Option	Description
Bit field Byte allocation (-BfaB[MS|LS])	By default, bit allocation in byte bitfields proceeds from the least significant bit to the most significant bit. This produces less code overhead in the case of partially- allocated byte bitfields. Options are: • MS: Most significant bit in byte first (left to right) • LS: Least significant bit in byte first (right to left)
Bit field gap limit (-BfaGapLimitBits)	Check to affect the maximum allowable number of gap bits. The bitfield allocation tries to avoid crossing a byte boundary whenever possible. To optimize accesses, the compiler may insert some padding or gap bits.
Bit field type size reduction	This option is configurable whether or not the compiler uses type-size reduction for bitfields. Type-size reduction means that the compiler can reduce the type of an int bitfield to a char bitfield if it fits into a character. This allows the compiler to allocate memory only for one byte instead of for an integer. Options are: • **Enabled** (-BfaTSRON) • **Disabled** (-BfaTSROFF)
Maximum load factor for switch tables (-100) (-CswMaxLF)	Allows changing the default strategy of the Compiler to use tables for switch statements; is only available if the compiler supports switch tables.
Minimum number of labels for switch tables (-CswMinLB)	Allows changing the default strategy of the Compiler using tables for switch statements; is only available if the compiler supports switch tables.
Minimum load factor for switch tables (100) (-CswMinLF)	Allows the Compiler to use tables for switch statements; is only available if the compiler supports switch tables.

Table 3.30 Tool Settings — HCS08 Compiler > Code Generation Options (*continued*)

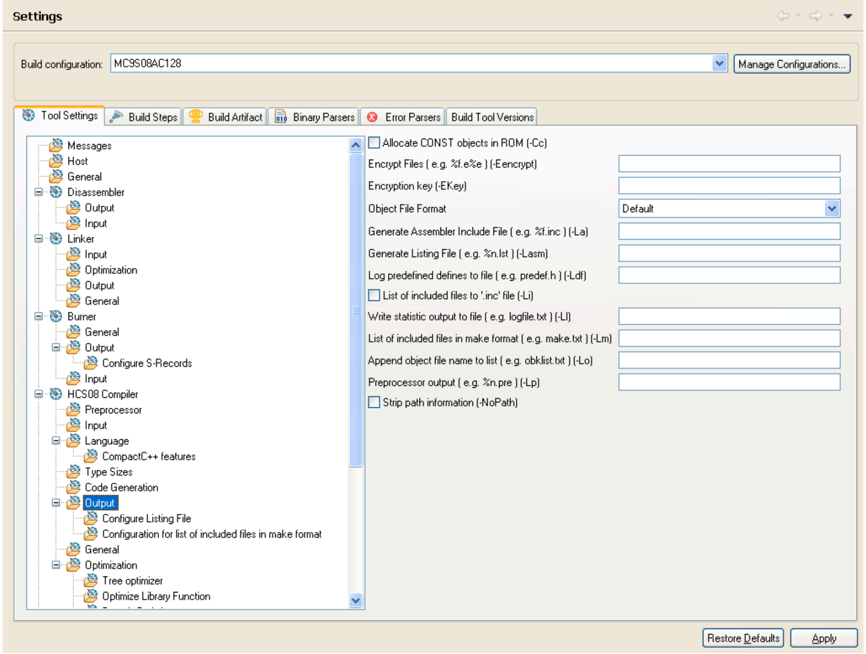
Option	Description
Minimum number of labels for switch search tables (-CswMinSLB)	Allows the Compiler to use tables for switch statements. Using a search table improves code density, but the execution time increases. Every time an entry in a search table must be found, all previous entries must be checked first. For a dense table, the right offset is computed and accessed. In addition, note that all backends implement search tables (if at all) by using a complex runtime routine. This may make debugging more complex.
Switch off code generation (-Cx)	Disables the code generation process of the Compiler. No object code is generated, though the Compiler performs a syntactical check of the source code. This allows a quick test if the Compiler accepts the source without errors.
Do not use CLR for volatile variables in the direct page (-NoClrVol)	Inhibits the use of CLR for volatile variables in the direct page. The CLR instruction on HC08 has a read cycle. This may lead to unwanted lateral effects (e.g. if the variable is mapped over a hardware register).
Qualifier for virtual table pointers (-Qvtp)	Using a virtual function in C++ requires an additional pointer to virtual function tables. The Compiler cannot access the pointer and generates the pointer in every class object when virtual function tables are associated. Options are: • None • Near • Far
Use IEEE32 for double (default is IEEE64)	Check to use doubles that are in IEEE32 instead of IEEE64 (default).
Assume HLI code saves modified registers	With this option set, the compiler assumes that registers touched in HLI are saved or restored in the HLI code as well. If this option is not set, the compiler saves or restores the H, X, and A registers.

HCS08 Compiler > Output

Use this panel to control how the compiler generates the output file, as well as error and warning messages. You can specify whether to allocate constant objects in ROM, generate debugging information, and strip file path information.

Figure 3.40 shows the **Optimization** panel.

Figure 3.36 Tool Settings — HCS08 Compiler > Output


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Allocate CONST objects in ROM (-Cc)
Encrypt Files (e.g. %f.e%e) (-Eencrypt)
Encryption key (-EKey)
Object File Format: Default
Generate Assembler Include File (e.g. %f.inc) (-La)
Generate Listing File (e.g. %n.lst) (-Lasm)
Log predefined defines to file (e.g. predef.h) (-Ldf)
☐ List of included files to '.inc' file (-Li)
Write statistic output to file (e.g. logfile.txt) (-Ll)
List of included files in make format (e.g. make.txt) (-Lm)
Append object file name to list (e.g. obklist.txt) (-Lo)
Preprocessor output (e.g. %n.pre) (-Lp)
☐ Strip path information (-NoPath)

Table 3.31 lists and describes the output options for HCS08 compiler.

Table 3.31 Tool Settings — HCS08 Compiler > Output Options

Option	Description
Allocate CONST objects in ROM (-Cc)	Check to enables the Compiler assign const objects into the ROM_VAR segment, which the parameter file assigns to a ROM section.
Encrypt Files (e.g. %f.e%e)(-Eencrypt)	Encrypts using the given key with the -Ekey: Encryption Key option.

Table 3.31 Tool Settings — HCS08 Compiler > Output Options (*continued*)

Option	Description
Encryption key (-Ekey)	Encrypt files with the given key number (-Eencrypt option).The default encryption key is 0. Using this default is not recommended.
Object File Format	Defines the format for the output file generated by the Assembler.
Generate Assembler Include File (e.g. %f.inc)(-La)	Enables the Compiler to generate an assembler include file when the CREATE_ASM_LISTING pragma occurs. The name of the created file is specified by this option. If no name is specified, a default of %f.inc is taken. To put the file into the directory specified by the TEXTPATH: Text File Path environment variable, use the option -la=%n.inc. The %f option already contains the path of the source file. When %f is used, the generated file is in the same directory as the source file. The content of all modifiers refers to the main input file and not to the actual header file. The main input file is the one specified on the command line.
Generate Listing File (e.g. %n.lst)(-Lasm)	Enables the Compiler to generate an assembler listing file directly. The Compiler also prints all assembler-generated instructions to this file. The option specifies the name of the file. If no name is specified, the Compiler takes a default of %n.lst. If the resulting filename contains no path information the Compiler uses the TEXTPATH: Text File Path environment variable. The syntax does not always conform with the inline assembler or the assembler syntax. Therefore, use this option only to review the generated code. It cannot currently be used to generate a file for assembly.

Table 3.31 Tool Settings — HCS08 Compiler > Output Options (*continued*)

Option	Description
Log predefined defines to file (e.g. predef.h)(-Ldf)	Enables the Compiler to generate a text file that contains a list of the compiler-defined #define. The default filename is predef.h, but may be changed (e.g., -Ldf="myfile.h"). The file is generated in the directory specified by the TEXTPATH: Text File Path environment variable. The defines written to this file depend on the actual Compiler option settings (e.g., type size settings or ANSI compliance). **Note:** The defines specified by the command line (-D: Macro Definition option) are not included. This option may be very useful for SQA. With this option it is possible to document every #define which was used to compile all sources. Note: This option only has an effect if a file is compiled. This option is unusable if you are not compiling a file.
List of included files to '.inc' file (-Li)	Enables the Compiler to generate a text file which contains a list of the #include files specified in the source. This text file shares the same name as the source file but with the extension, *.inc. The files are stored in the path specified by the TEXTPATH: Text File Path environment variable. The generated file may be used in make files.

Table 3.31 Tool Settings — HCS08 Compiler > Output Options (*continued*)

Option	Description
Write static output to file (e.g. logfile.txt)(-Ll)	Enables the Compiler append statistical information about the compilation session to the specified file. The information includes Compiler options, code size (in bytes), stack usage (in bytes) and compilation time (in seconds) for each procedure of the compiled file. The Compiler appends the information to the specified filename (or the file make.txt, if no argument given). Set the TEXTPATH: Text File Path environment variable to store the file into the path specified by the environment variable. Otherwise the Compiler stores the file in the current directory.
List of included files in make format (e.g. make.txt)(-Lm)	This option causes the Compiler to generate a text file which contains a list of the #include files specified in the source. The generated list is in a make format. The -Lm option is useful when creating make files. The output from several source files may be copied and grouped into one make file. The generated list is in the make format. The filename does not include the path. After each entry, an empty line is added. The information is appended to the specified filename (or the make.txt file, if no argument is given).
Append object file name to list (e.g.obklist.txt)(-Lo)	This option causes the Compiler to append the object filename to the list in the specified file.The information is appended to the specified filename (or the file make.txt file, if no argument given).

Table 3.31 Tool Settings — HCS08 Compiler > Output Options (*continued*)

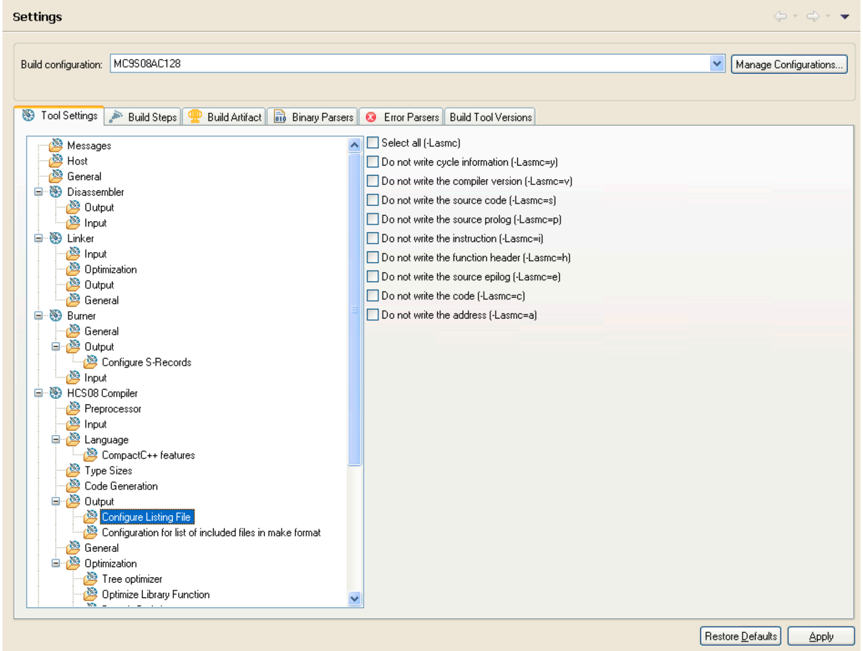
Option	Description
Processor output (e.g. %n.pre)(-Lp)	This option causes the Compiler to generate a text file which contains the preprocessor's output. If no filename is specified, the text file shares the same name as the source file but with the extension, *.PRE (%n.pre). The TEXTPATH environment variable is used to store the preprocessor file.
Strip path information (-NoPath)	Check to enable the compiler remove both unreferenced path reference from your program. This reduces your program's memory footprint.

HCS08 Compiler > Output > Configure Listing File

Use this panel to configure the listing file options of the HCS08 compiler.

Figure 3.37 shows **Compiler > Output > Configure Listing File** panel.

Figure 3.37 Tool Settings — HCS08 Compiler > Output > Configure Listing File


Settings
Build configuration: MC9S08AC128
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Restore Defaults
Apply

☐ Select all (-Lasmc)
☐ Do not write cycle information (-Lasmc=y)
☐ Do not write the compiler version (-Lasmc=v)
☐ Do not write the source code (-Lasmc=s)
☐ Do not write the source prolog (-Lasmc=p)
☐ Do not write the instruction (-Lasmc=i)
☐ Do not write the function header (-Lasmc=h)
☐ Do not write the source epilog (-Lasmc=e)
☐ Do not write the code (-Lasmc=c)
☐ Do not write the address (-Lasmc=a)

Table 3.32 lists and describes the **Configure Listing File** options for the HC(S)08 Compiler.

Table 3.32 Tool Settings — HCS08 Compiler > Output > Configure Listing File Options

Option	Description
Select all (-Lasmc)	This option configures the output format of the listing file generated with the *Generate Listing File* option. The addresses, the hex bytes, and the instructions are selectively switched off.
Do not write cycle information (-Lasmc=y)	This option switches off the cycle information from the output format of the listing file.
Do not write the compiler version (-Lasmc=v)	This option switches off the compiler version from the output format of the listing file.
Do not write the source code (-Lasmc=s)	This option switches off the source code from the output format of the listing file.

Table 3.32 Tool Settings — HCS08 Compiler > Output > Configure Listing File Options

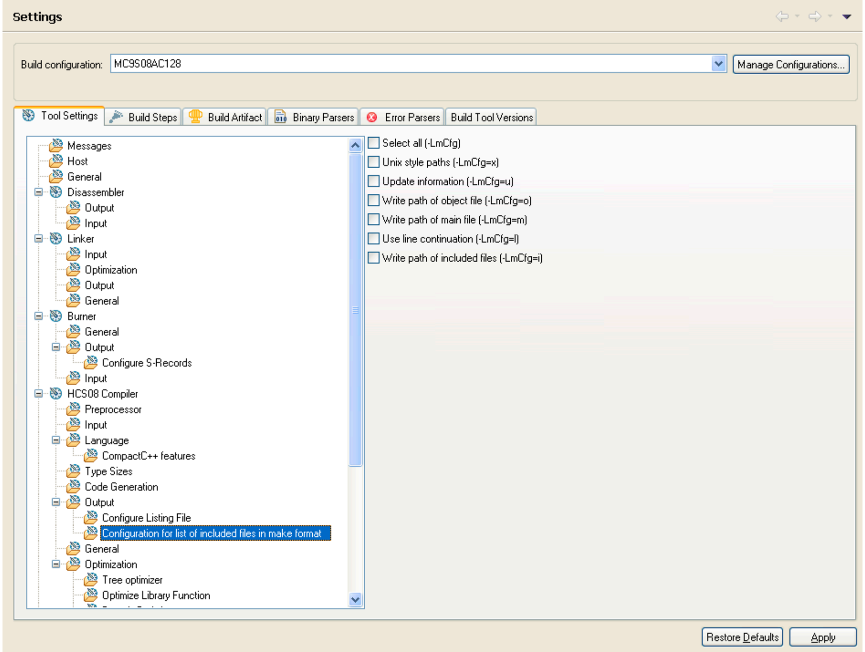
Option	Description
Do not write the source prolog (-Lasmc=p)	This option switches off the source prolog from the output format of the listing file.
Do not write the instruction (-Lasmc=i)	This option switches off the instruction from the output format of the listing file.
Do not write the function header (-Lasmc=h)	This option switches off the function header from the output format of the listing file.
Do not write the source epilog (-Lasmc=e)	This option switches off the source epilog from the output format of the listing file.
Do not write the code (-Lasmc=c)	This option switches off the code from the output format of the listing file.
Do not write the address (-Lasmc=a)	This option switches off the address from the output format of the listing file.

HCS08 Compiler > Output > Configuration for list of included files in make format

Use this panel to configure the list of included files in make format of the HCS08 compiler.

Figure 3.38 shows **Output > Configuration for list of included files in make format** panel.

Figure 3.38 Tool Settings — HCS08 Compiler > Output > Configuration for List of Included Files in Make Format


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Select all (-LmCfg)
☐ Unix style paths (-LmCfg=x)
☐ Update information (-LmCfg=u)
☐ Write path of object file (-LmCfg=o)
☐ Write path of main file (-LmCfg=m)
☐ Use line continuation (-LmCfg=l)
☐ Write path of included files (-LmCfg=i)

Table 3.33 lists and describes the **Configuration for list of included files in make format** options for HC(S)08 compiler

Table 3.33 Tool Settings — HCS08 Compiler > Output > Configuration for List of Included Files in Make Format Options

Option	Description
Select all (-LmCfg)	This option is used when configuring the *List of Included Files in Make Format (-Lm)* option. The -LmCfg option is operative only if the -Lm option is also used. The -Lm option produces the ‘dependency’ information for a make file.
Unix style paths (-LmCfg=x)	Use this option to writes the path names in Unix style.

Table 3.33 Tool Settings — HCS08 Compiler > Output > Configuration for List of Included Files in Make Format Options

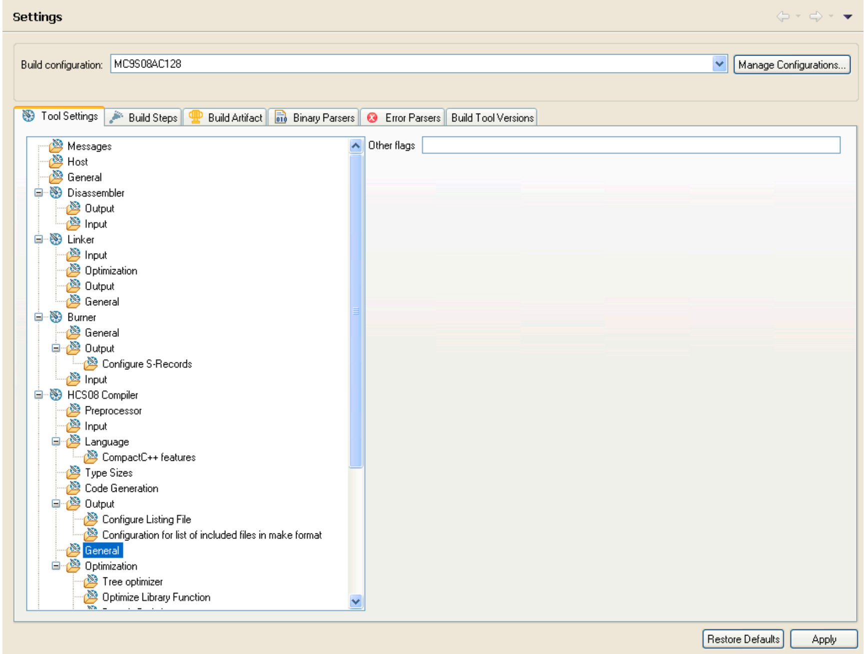
Option	Description
Update information (-LmCfg=u)	This option updates the information in the output file. If the file does not exist, the Compiler creates the file. If the file exists and the current information is not yet in the file, the Compiler appends the information to the file. If the information is already present, the Compiler updates the information. This allows you to specify this suboption for each compilation ensuring that the make dependency file is always up to date.
Write path of object file (-LmCfg=o)	This option writes the full name of the target object file.
Write path of main file (-LmCfg=m)	This option writes the full path of the compiled file. This is necessary when there are files with the same name in different directories.
Use line continuation (-LmCfg=l)	This option uses line continuation for each single entry in the dependency list. This improves readability.
Write path of included files (-LmCfg=i)	This option writes the full path of all included files in the dependency list.

HCS08 Compiler > General

Use this panel to specify other flags for the HC(S)08 Compiler to use.

Figure 3.39 shows the **General** panel.

Figure 3.39 Tool Settings — HCS08 Compiler > General



Build configuration: MC9S08AC128

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Other flags
Restore Defaults
Apply

Table 3.34 lists and describes the **General** options for HC(S)08 compiler.

Table 3.34 Tool Settings — HC08 Compiler > General Options

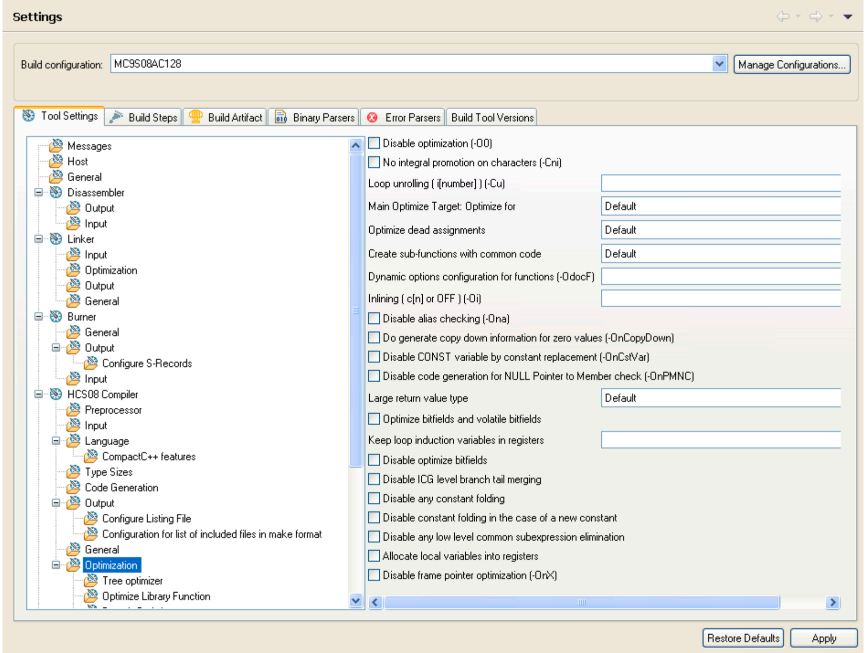
Option	Description
Other flags	Specify additional command line options for the compiler; type in custom flags that are not otherwise available in the UI.

HCS08 Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.40 shows the **Optimization** panel.

Figure 3.40 Tool Settings — HCS08 Compiler > Optimization


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Disable optimization (-O0)
☐ No integral promotion on characters (-Cni)
Loop unrolling (i[number]) (-Cu)
Main Optimize Target: Optimize for: Default
Optimize dead assignments: Default
Create sub-functions with common code: Default
Dynamic options configuration for functions (-OdocF)
Inlining (c[n] or OFF) (-Oi)
☐ Disable alias checking (-Ona)
☐ Do generate copy down information for zero values (-OnCopyDown)
☐ Disable CONST variable by constant replacement (-OnCstVar)
☐ Disable code generation for NULL Pointer to Member check (-OnPMNC)
Large return value type: Default
☐ Optimize bitfields and volatile bitfields
Keep loop induction variables in registers
☐ Disable optimize bitfields
☐ Disable ICG level branch tail merging
☐ Disable any constant folding
☐ Disable constant folding in the case of a new constant
☐ Disable any low level common subexpression elimination
☐ Allocate local variables into registers
☐ Disable frame pointer optimization (-OnX)

Table 3.35 lists and describes the **Optimization** options for HC(S)08 compiler.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options

Option	Description
Disable optimization (-O0)	Disables all optimizations.
No integral promotion on characters (-Cni)	Enhances character operation code density by omitting integral promotion. This option enables behavior that is not ANSI-C compliant. Code generated with this option set does not conform to ANSI standards. Code compiled with this option is not portable. Using this option is not recommended in most cases.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Loop unrolling (i[number]) (-Cu)	Enables loop unrolling with the following restrictions: • Only simple for statements are unrolled, for example, for (i=0; i<10; i++) • Initialization and test of the loop counter must be done with a constant. • Only <, >, <=, >= are permitted in a condition. • Only ++ or -- are allowed for the loop variable increment or decrement. • The loop counter must be integral. • No change of the loop counter is allowed within the loop. • The loop counter must not be used on the left side of an assignment. • No address operator (&) is allowed on the loop counter within the loop. • Only small loops are unrolled: Loops with few statements within the loop. Loops with fewer than 16 increments or decrements of the loop counter. The bound may be changed with the optional argument = i<number>. The -Cu=i20 option unrolls loops with a maximum of 20 iterations.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Main Optimize Target: Optimize for	There are various points where the Compiler has to select between two possibilities: it can either generate fast, but large code, or small but slower code. The Compiler generally optimizes on code size. It often has to decide between a runtime routine or an expanded code. The programmer can decide whether to select between the slower and shorter or the faster and longer code sequence by setting a command line switch. • The **Code Size (-Os)** option directs the Compiler to optimize the code for smaller code size. The Compiler trades faster-larger code for slower-smaller code. • The **Execution Time (-Ot)** option directs the Compiler to optimize the code for faster execution time. The Compiler replaces slower/ smaller code with faster/larger code. This option only affects some special code sequences. This option has to be set together with other optimization options (e.g., register optimization) to get best results.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Optimize dead assignments	Optimizes dead assignments. The Compiler removes assignments to unused local variables. There are three possible settings for this option: • **always (even if HLI present in function):** Always optimize dead assignments (even if HLI is present in current function). The Compiler does not consider inline assembler accesses. **Note:** This option is unsafe when inline assembler code contains accesses to local variables. • **yes, but never if HLI present in function**: No optimization occurs. This generates the best possible debug information, and produces larger and slower code. • **never**: Optimize dead assignments if HLI is not present in the current function.
Create sub-functions with common code	Performs the reverse of inlining. It detects common code parts in the generated code. The Compiler moves the common code to a different place and replaces all occurrences with a JSR to the moved code. At the end of the common code, the Compiler inserts an RTS instruction. The Compiler increases all SP uses by an address size. This optimization takes care of stack allocation, control flow, and of functions having arguments on the stack. Inline assembler code is never treated as common code. Options are: • **Default** • **Disable (-Onf)** • **Off (-Of)**

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Dynamic options configuration for functions (-OdocF)	Allows the Compiler to select from a set of options to reach the smallest code size for every function. Without this feature, you must set fixed Compiler switches over the whole compilation unit. With this feature, the Compiler finds the best option combination from a user-defined set for every function.
Inlining (C[n] or OFF) (-Oi)	Enables inline expansion. If there is a #pragma INLINE before a function definition, all calls of this function are replaced by the code of this function, if possible. Using the -Oi=c0 option switches off inlining. Functions marked with the #pragma INLINE are still inlined. To disable inlining, use the -Oi=OFF option.
Disable alias checking (-Ona)	Prevents the Compiler from redefining these variables, which allows you to reuse already-loaded variables or equivalent constants. Use this option only when you are sure no real writes of aliases to a variable memory location will occur.
Do generate copy down information for zero values (-OnCopyDown)	Using this option, the compiler does not generate a copy down for i. The initialization with zero optimization shown for the arr array only works in the HIWARE format. The ELF format requires initializing the whole array to zero.
Disable CONST variable by constant replacement (-OnCstVar)	Allows you to switch OFF the replacement of CONST variable by the constant value.
Disable code generation for NULL Pointer to Member check (-OnPMNC)	Before assigning a pointer to a member in C++, you must ensure that the pointer to the member is not NULL in order to generate correct and safe code. In embedded systems development, the difficulty becomes generating the denser code while avoiding overhead whenever possible (this NULL check code is a good example). This option enables you to switch off the code generation for the NULL check.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Large return value type	Compiler supports this option even though returning a 'large' return value may be not as efficient as using an additional pointer. The Compiler introduces an additional parameter for the return value if the return value cannot be passed in registers. Options are: • Default • Large return value pointer, always with temporary (-Rpt) • Large return value pointer and temporary elimination (-Rpe)
Optimize bitfields and volatile bitfields	Use this option to optimize bitfields and volatile bitfields. The compiler changes the access order or combines many accesses into one, even if the bitfields are declared as volatile.
Keep loop induction variables in registers	Limits the number of loop induction variables the Compiler keeps in registers. Specify any number down to zero (no loop induction variables). The compiler reads and writes loop induction variables within the loop (for example, loop counter), and attempts to keep the variables in registers to reduce execution time and code size. The Compiler takes the optimal number (code density) when this option is not specified. Specifying a high number of loop induction variables may increase code size, particularly for spill and merge code.
Disable optimize bitfields	Prevents the Compiler from combining sequences of bitfield assignments containing constants. This simplifies debugging and makes the code more readable.
Disable ICG level branch tail merging	Switches the ICG level branch tail merging off. This simplifies debugging and produces more readable code.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Disable any constant folding	Prevents the Compiler from folding constants over statement boundaries. All arithmetical operations are coded. This option must be set when using the library functions setjmp() and longjmp(), or the Compiler makes wrong assumptions.
Disable constant folding in the case of a new constant	This option prevents the Compiler from folding constants when the resulting constant is new. The option affects only those processors where constants are difficult to load (e.g., RISC processors). On other processors this option makes no change.
Disable any low level common subexpression elimination	Prevents the Compiler from reusing common subexpressions, such as array indexes and array base addresses. The code size may increase. The low-level CSE does not have the alias problems of the frontend CSE and is therefore switched on by default. The two CSE optimizations do not cover the same cases. The low-level CSE has a finer granularity but does not handle all cases of the frontend CSE. Use this option only to generate more readable code for debugging.

Table 3.35 Tool Settings — HCS08 Compiler > Optimization Options (*continued*)

Option	Description
Allocate local variables into registers	Allocates local variables (char or int) in registers. The number of local variables allocated in registers depends on the number of available registers. Use this option when using variables as loop counters or switch selectors or when the processor requires register operands for multiple operations (for example, RISC processors). Compiling with this option may increase your code size (spill and merge code). This optimization may increase code complexity when using High-Level Languages, making debugging more difficult.
Disable frame pointer optimization (-OnX)	Prevents the Compiler from converting stack pointer-relative accesses into X-relative accesses. The frame optimizer tries to convert all SP-relative accesses (local variables, spills) into shorter and faster X-relative accesses. In addition, the Compiler traces the value of H:X and removes useless TSX and AIX instructions. Using -OnX to switch the frame optimizer off facilitates debugging.

HCS08 Compiler > Optimization > Tree optimizer

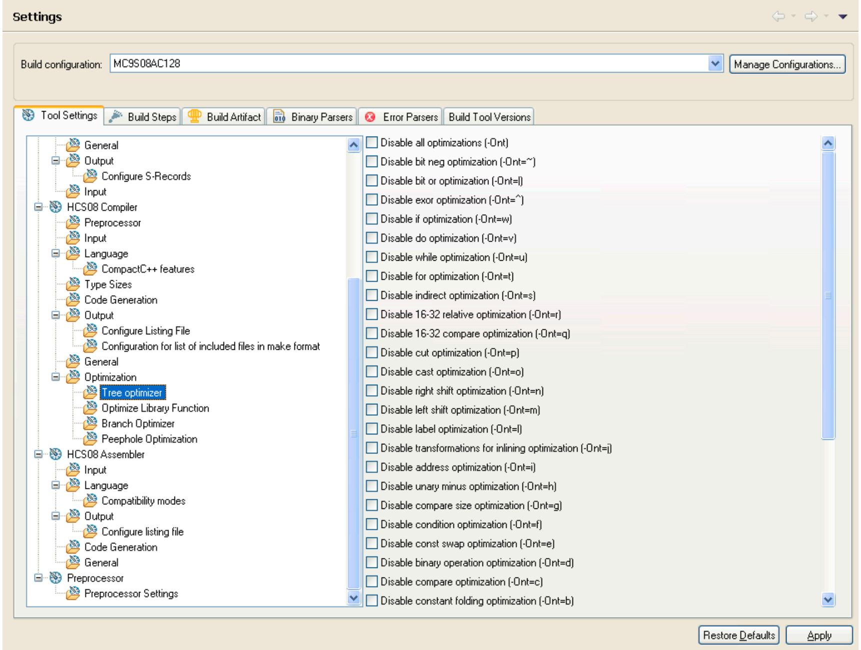
The Compiler contains a special optimizer which optimizes the internal tree data structure. This tree data structure holds the semantic of the program and represents the parsed statements and expressions.

This option disables the tree optimizer. This may be useful for debugging and for forcing the Compiler to produce ‘straightforward’ code.

Use this panel to configure the tree optimizer options for the HCS08 compiler.

Figure 3.41 shows the **Compiler > Optimization > Tree optimizer** panel.

Figure 3.41 Tool Settings — HCS08 Compiler > Optimization > Tree optimizer


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Disable all optimizations (-Ont)
☐ Disable bit neg optimization (-Ont=~)
☐ Disable bit or optimization (-Ont=|)
☐ Disable exor optimization (-Ont=^)
☐ Disable if optimization (-Ont=w)
☐ Disable do optimization (-Ont=v)
☐ Disable while optimization (-Ont=u)
☐ Disable for optimization (-Ont=t)
☐ Disable indirect optimization (-Ont=s)
☐ Disable 16-32 relative optimization (-Ont=r)
☐ Disable 16-32 compare optimization (-Ont=q)
☐ Disable cut optimization (-Ont=p)
☐ Disable cast optimization (-Ont=o)
☐ Disable right shift optimization (-Ont=n)
☐ Disable left shift optimization (-Ont=m)
☐ Disable label optimization (-Ont=l)
☐ Disable transformations for inlining optimization (-Ont=j)
☐ Disable address optimization (-Ont=i)
☐ Disable unary minus optimization (-Ont=h)
☐ Disable compare size optimization (-Ont=g)
☐ Disable condition optimization (-Ont=f)
☐ Disable const swap optimization (-Ont=e)
☐ Disable binary operation optimization (-Ont=d)
☐ Disable compare optimization (-Ont=c)
☐ Disable constant folding optimization (-Ont=b)

Table 3.36 lists and describes the **Tree optimizer** options for HC(S)08 compiler.

Table 3.36 Tool Settings — HCS08 Compiler > Optimization > Tree optimizer

Option	Description
Disable all optimizations (-Ont)	Disable all the optimizations.
Disable bit neg optimization (-Ont=~)	Disable optimization of '~~i' into 'i'.
Disable bit or optimization (-Ont=|)	Disable optimization of 'i|0xffff' into '0xffff'.
Disable exor optimization (-Ont=^)	Disable optimization of 'i^0' into 'i'.
Disable if optimization (-Ont=w)	Disable optimization of 'if (1) i = 0;' into 'i = 0;'.
Disable do optimization (-Ont=v)	Disable optimization of 'do ... while(0) into '...'.
Disable while optimization (-Ont=u)	Disable optimization of 'while(1) ...;' into '...;'.

Table 3.36 Tool Settings — HCS08 Compiler > Optimization > Tree optimizer

Option	Description
Disable for optimization (-Ont=t)	Disable optimization of ‘for(;;) ...’ into ‘while(1) ...’.
Disable indirect optimization (-Ont=s)	Disable optimization of ‘*&i’ into ‘i’.
Disable 16-32 relative optimization (-Ont=r)	Disable optimization of ‘L<=4’ into 16-bit compares if 16-bit compares are better.
Disable 16-32 compare optimization (-Ont=q)	Reduction of long compares into int compares if int compares are better: (-Ont=q to disable it).
Disable cut optimization (-Ont=p)	Disable optimization of ‘(char)(long)i’ into ‘(char)i’.
Disable cast optimization (-Ont=o)	Disable optimization of ‘(short)(int)L’ into ‘(short)L’ if short and int have the same size.
Disable right shift optimization (-Ont=n)	Optimization of shift optimizations (<<, -Ont=n to disable it)
Disable left shift optimization (-Ont=m)	Optimization of shift optimizations (>>, -Ont=m to disable it)
Disable label optimization (-Ont=l)	Disable optimization removal of labels if not used.
Disable transformations for inlining optimization (-Ont=j)	This optimization transforms the syntax tree into an equivalent form in which more inlining cases can be done. This option only has an effect when inlining is enabled.
Disable address optimization (-Ont=i)	Disable optimization of ‘&*p’ into ‘p’.
Disable unary minus optimization (-Ont=h)	Disable optimization of ‘-(-i)’ into ‘i’.
Disable compare size optimization (-Ont=g)	
Disable condition optimization (-Ont=f)	Disable optimization of ‘(a==0)’ into ‘(!a)’.
Disable const swap optimization (-Ont=e)	Disable optimization of ‘2*i’ into ‘i*2’.
Disable binary operation optimization (-Ont=d)	Disable optimization of ‘us & ui’ into ‘us & (unsigned short) ui’.

Table 3.36 Tool Settings — HCS08 Compiler > Optimization > Tree optimizer

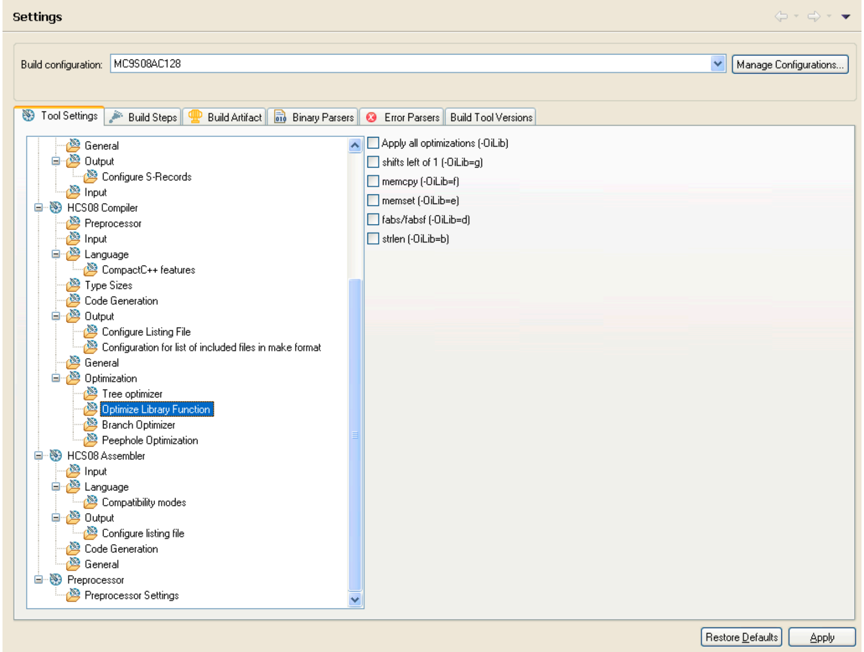
Option	Description
Disable compare optimization (-Ont=c)	Disable optimization of ‘if ((long)i)’ into ‘if (i)’.
Disable constant folding optimization (-Ont=b)	Disable optimization of ‘3+7’ into ‘10’.
Disable statement optimization (-Ont=a)	Disable optimization of last statement in function if result is not used.
Disable test optimization (-Ont=?)	Disable optimization of ‘i = (int)(cond ? L1:L2);’ into ‘i = cond ? (int)L1:(int)L2;’.
Disable assign optimization (-Ont=9)	Disable optimization of ‘i=i;’.
Disable switch optimization (-Ont=8)	Disable optimization of empty switch statement.
Disable extend optimization (-Ont=7)	Disable optimization of ‘(long)(char)L’ into ‘L’.
Disable or optimization (-Ont=1)	Disable optimization of ‘a || 0’ into ‘a’.
Disable and optimization (-Ont=0)	Disable optimization of ‘a && 1’ into ‘a’.
Disable div optimization (-Ont=/)	Disable optimization of ‘a/1’ into ‘a’.
Disable minus optimization (-Ont=-)	Disable optimization of ‘a-0’ into ‘a’.
Disable plus optimization (-Ont=+)	Disable optimization of ‘a+0’ into ‘a’.
Disable mul optimization (-Ont=*)	Disable optimization of ‘a*1’ into ‘a’.
Disable bit and optimization (-Ont=)	Disable optimization of ‘a&0’ into ‘0’.
Disable mod optimization (-Ont=%)	Disable optimization of ‘a%1’ into ‘0’.

HCS08 Compiler > Optimization > Optimize Library Function

This option enables the compiler to optimize specific known library functions to reduce execution time. The Compiler frequently uses small functions such as strcpy(), strcmp(), and so forth. Use this panel to configure the optimize library function options for the HCS08 compiler.

Figure 3.42 shows the **Compiler > Optimization > Optimize Library Function** panel.

Figure 3.42 Tool Settings — HCS08 Compiler > Optimization > Optimize Library Function



Build configuration: MC9S08AC128

☐ Apply all optimizations (-OiLib)
☐ shifts left of 1 (-OiLib=g)
☐ memcpy (-OiLib=f)
☐ memset (-OiLib=e)
☐ fabs/fabsf (-OiLib=d)
☐ strlen (-OiLib=b)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.37 lists and describes **Optimize Library Function** options for HC(S)08 compiler.

Table 3.37 Tool Settings — HCS08 Compiler > Optimization > Optimize Library Function

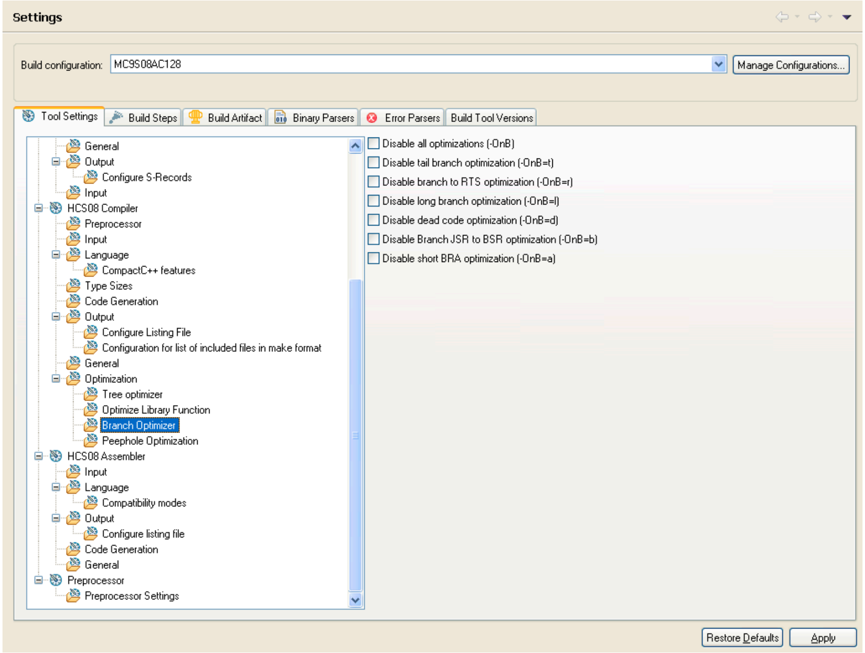
Option	Description
Apply all optimizations (-OiLib)	This option applies all the optimizations.
shifts left of 1 (-OiLib=g)	This option replace shifts left of 1 by array lookup.
memcpy (-OiLib=f)	This option inline calls to the memcpy() function.
memset (-OiLib=e)	This option inline calls to the memset() function.
fabs/fabsf (-OiLib=d)	This option inline calls to the fabs() or fabsf() functions.
strlen (-OiLib=b)	This option inline calls to the strlen() function.

HCS08 Compiler > Optimization > Branch Optimizer

Use this panel to specify the branch optimizer options of the HCS08 compiler.

Figure 3.43 shows the **Optimization** > **Branch Optimizer** panel.

Figure 3.43 Tool Settings — HCS08 Compiler > Optimization > Branch Optimizer



Settings
Build configuration:
MC9S08AC128
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

☐ Disable all optimizations (-OnB)
☐ Disable tail branch optimization (-OnB=t)
☐ Disable branch to RTS optimization (-OnB=r)
☐ Disable long branch optimization (-OnB=l)
☐ Disable dead code optimization (-OnB=d)
☐ Disable Branch JSR to BSR optimization (-OnB=b)
☐ Disable short BRA optimization (-OnB=a)

Table 3.38 lists and describes the **Branch Optimizer** options for HC(S)08 compiler.

Table 3.38 Tool Settings — HCS08 Compiler > Optimization > Branch Optimizer

Option	Description
Disable all optimizations (-OnB)	With this option, all low-level branch optimizations are disabled.
Disable tail branch optimization (-OnB=t)	Disable Branch tail optimization
Disable branch to RTS optimization (-OnB=r)	Disable Branch to RTS optimization
Disable long branch optimization (-OnB=l)	Disable long branch optimization

Table 3.38 Tool Settings — HCS08 Compiler > Optimization > Branch Optimizer

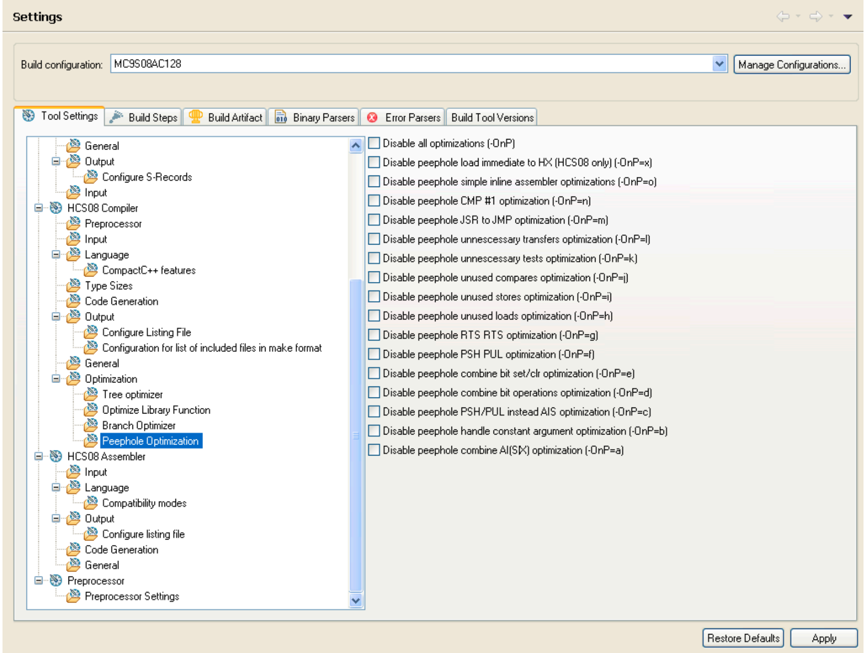
Option	Description
Disable dead code optimization (-OnB=d)	Disable dead code optimization
Disable Branch JSR to BSR optimization (-OnB=b)	Disable Branch JSR to BSR optimization
Disable short BRA optimization (-OnB=a)	Disable short BRA optimization

HCS08 Compiler > Optimization > Peephole Optimization

Use this panel to configure peephole optimization for the HC(S)08 Compiler.

Figure 3.44 shows the **Peephole Optimization** panel.

Figure 3.44 Tool Settings — HCS08 Compiler > Optimization > Peephole Optimization


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Disable all optimizations (-OnP)
☐ Disable peephole load immediate to HX (HCS08 only) (-OnP=x)
☐ Disable peephole simple inline assembler optimizations (-OnP=o)
☐ Disable peephole CMP #1 optimization (-OnP=n)
☐ Disable peephole JSR to JMP optimization (-OnP=m)
☐ Disable peephole unnescessary transfers optimization (-OnP=l)
☐ Disable peephole unnescessary tests optimization (-OnP=k)
☐ Disable peephole unused compares optimization (-OnP=j)
☐ Disable peephole unused stores optimization (-OnP=i)
☐ Disable peephole unused loads optimization (-OnP=h)
☐ Disable peephole RTS RTS optimization (-OnP=g)
☐ Disable peephole PSH PUL optimization (-OnP=f)
☐ Disable peephole combine bit set/clr optimization (-OnP=e)
☐ Disable peephole combine bit operations optimization (-OnP=d)
☐ Disable peephole PSH/PUL instead AIS optimization (-OnP=c)
☐ Disable peephole handle constant argument optimization (-OnP=b)
☐ Disable peephole combine AI(S|X) optimization (-OnP=a)

Table 3.39 lists and describes the **Peephole Otimization** options for HC(S)08 compiler.

Table 3.39 Tool Settings — HCS08 Compiler > Optimization > Peephole Optimization

Option	Description
Disable all optimizations (-OnP)	If -OnP is specified, the Compiler disables the whole peephole optimizer.
Disable peephole load immediate to HX (HCS08 only) (-OnP=x)	Disable peephole load immediate to HX (HCS08 only).
Disable peephole simple inline assembler optimizations (-OnP=o)	Disable peephole simple inline assembler optimizations.
Disable peephole CMP #1 optimization (-OnP=n)	Disable peephole CMP #1 optimization.
Disable peephole JSR to JMP optimization (-OnP=m)	Disable peephole JSR to JMP optimization.
Disable peephole unnescessary transfers optimization (-OnP=l)	Disable peephole unnescessary transfers optimization.
Disable peephole unnescessary tests optimization (-OnP=k)	Disable peephole unnescessary tests optimization.
Disable peephole unused compares optimization (-OnP=j)	Disable peephole unused compares optimization.
Disable peephole unused stores optimization (-OnP=i)	Disable peephole unused stores optimization.
Disable peephole unused loads optimization (-OnP=h)	Disable peephole unused loads optimization.
Disable peephole RTS RTS optimization (-OnP=g)	Disable peephole RTS RTS optimization.
Disable peephole PSH PUL optimization (-OnP=f)	Disable peephole PSH PUL optimization.
Disable peephole combine bit set/clr optimization (-OnP=e)	Disable peephole combine bit set/clr optimization.
Disable peephole combine bit operations optimization (-OnP=d)	Disable peephole combine bit operations optimization.
Disable peephole PSH/PUL instead AIS optimization (-OnP=c)	Disable peephole PSH/PUL instead AIS optimization.

Table 3.39 Tool Settings — HCS08 Compiler > Optimization > Peephole Optimization

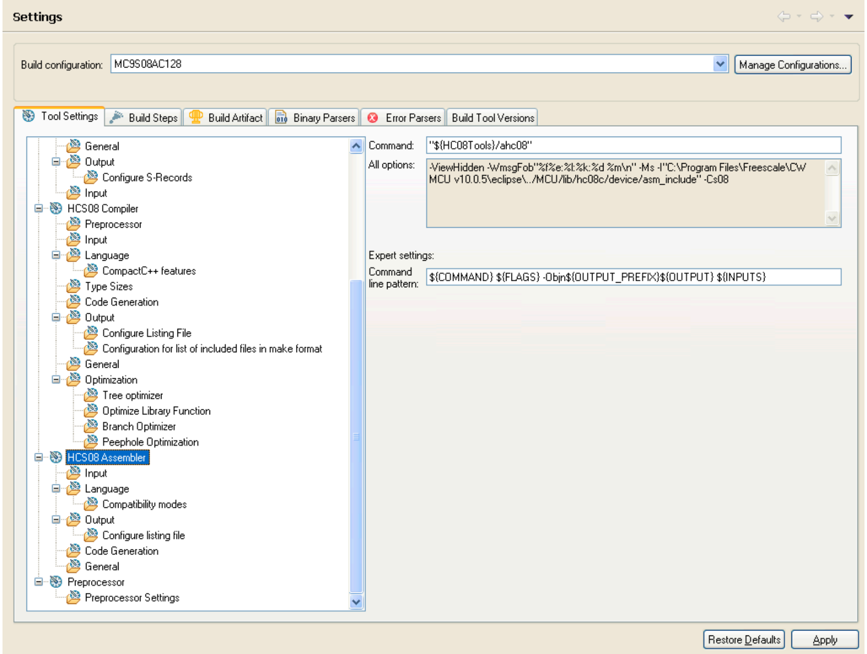
Option	Description
Disable peephole handle constant argument optimization (-OnP=b)	Disable peephole handle constant argument optimization.
Disable peephole combine AI(SIX) optimization (-OnP=a)	Disable peephole combine AI(SIX) optimization.

HCS08 Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler.

Figure 3.45 shows the **Assembler** settings.

Figure 3.45 Tool Settings — Assembler


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Expert settings:
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: "${HC08Tools}/ahc08"

All options: -ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -Ms -I"C:\Program Files\Freescale\CW MCU v10.0.5\eclipse\../MCU/lib/hc08c/device/asm_include" -Cs08

Command line pattern: ${COMMAND} ${FLAGS} -Objn${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.40 lists and describes the assembler options for HCS08.

Table 3.40 Tool Settings — Assembler Options

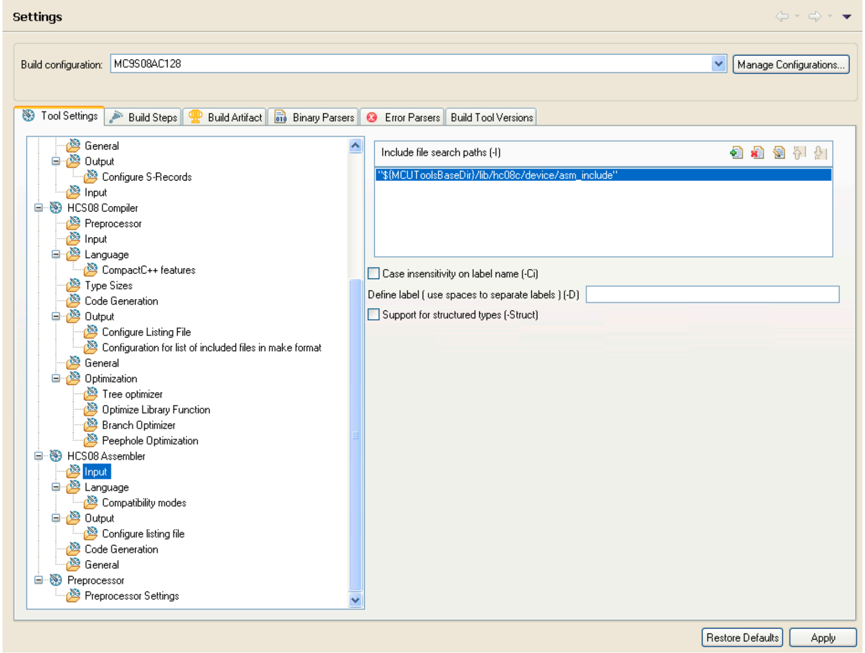
Option	Description
Command	Shows the location of the assembler executable file.
All options	Shows the actual command line the assembler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS}-Objn${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

HCS08 Assembler > Input

Use this panel to specify file search paths and any additional include files the **HCS08 Assembler** should use. You can specify multiple search paths and the order in which you want to perform the search.

Figure 3.46 shows the **Input** panel.

Figure 3.46 Tool Settings — HCS08 Assembler > Input


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

Include file search paths (-I)

"${MCUToolsBaseDir}/lib/hc08c/device/asm_include"

☐ Case insensitivity on label name (-Ci)

Define label (use spaces to separate labels) (-D)

☐ Support for structured types (-Struct)

Table 3.41 lists and describes the input options for HCS08 Assembler.

Table 3.41 Tool Settings — HCS08 Assembler > Input Options

Option	Description
Include File Search Paths (-I)	Lists the included file search paths.
Case sensitivity or label name (-Ci)	Check to make the label names case sensitive.
Define label (Use spaces to separate labels) (-D)	Define labels that have to be included in the RS08 assembler input.
Support for structured types (-Struct)	Check to include the support for structured types.

Table 3.42 lists and describes the toolbar buttons that help work with the file search paths.

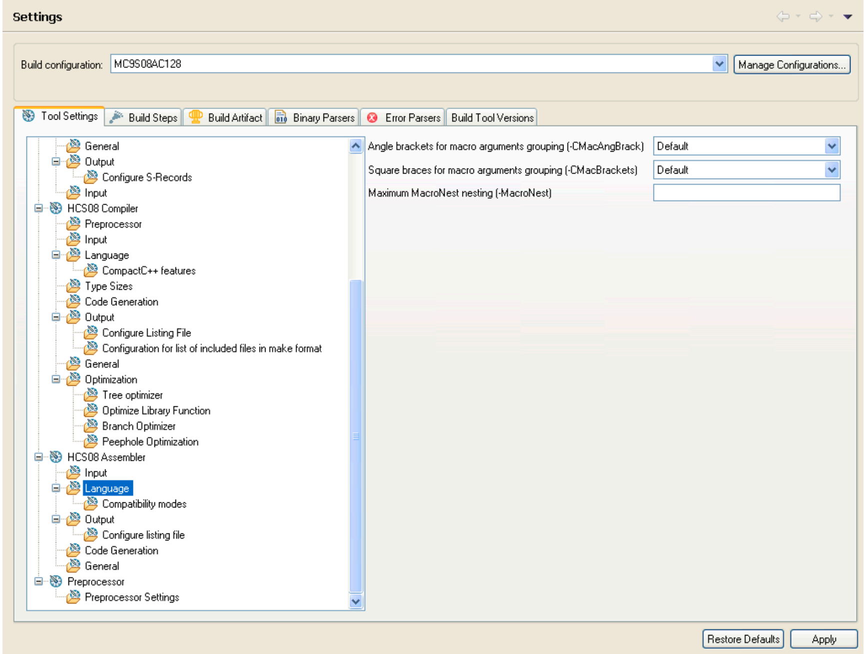
Table 3.42 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.14) and specify the file search path.
✖	**Delete** — Click to delete the selected file search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.15) and update the selected object file search path.
⇧	**Move up** — Click to move the selected file search path one position higher in the list.
⇩	**Move down** — Click to move the selected file search path one position lower in the list.

HCS08 Assembler > Language

Use this panel to specify code- and symbol-generation options for the HCS08 Compiler. Figure 3.47 shows the **Language** panel.

Figure 3.47 Tool Settings — HCS08 Assembler > Language


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

Angle brackets for macro arguments grouping (-CMacAngBrack)	Default
Square braces for macro arguments grouping (-CMacBrackets)	Default
Maximum MacroNest nesting (-MacroNest)	

Table 3.43 lists and describes the language options for HCS08 Assembler.

Table 3.43 Tool Settings — HCS08 Assembler > Language Options

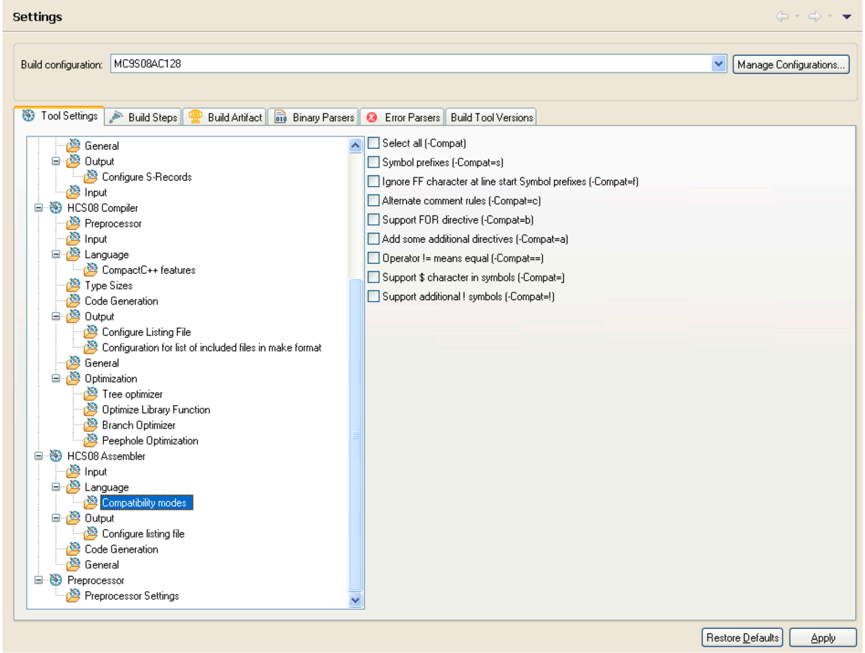
Option	Description
Angle brackets for macro arguments grouping (-CMacAngBrack)	Controls whether the < > syntax for macro invocation argument grouping is available. When it is disabled, the Assembler does not recognize the special meaning for < in the macro invocation context. There are cases where the angle brackets are ambiguous. In new code, use the [? ?] syntax instead. Options are: • Allow • Disallow
Square braces for macro arguments grouping (-CMacBrackets)	Controls the availability of the [? ?] syntax for macro invocation argument grouping. When it is disabled, the Assembler does not recognize the special meaning for [? in the macro invocation context. Options are: • Allow • Disallow
Maximum MacroNest nesting (-MacroNest)	Controls how deep macros calls can be nested. Its main purpose is to avoid endless recursive macro invocations.

HCS08 Assembler > Language > Compatibility modes

cghghkhlghujkhgjg

Figure 3.49 shows the **Output** panel.

Figure 3.48 Tool Settings — HCS08 Assembler > Language > Compatibility modes


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Select all (-Compat)
☐ Symbol prefixes (-Compat=s)
☐ Ignore FF character at line start Symbol prefixes (-Compat=f)
☐ Alternate comment rules (-Compat=c)
☐ Support FOR directive (-Compat=b)
☐ Add some additional directives (-Compat=a)
☐ Operator != means equal (-Compat==)
☐ Support $ character in symbols (-Compat=)
☐ Support additional ! symbols (-Compat=!)

Table 3.44 lists and describes the compatibility mode options for HCS08 Assembler.

Table 3.44 Tool Settings — HCS08 Assembler > Language > Compatibility mode Options

Option	Description
Select all (-Compat)	
Symbol prefixes (-Compat=s)	With this suboption, the Assembler accepts "pgz:" and "byte:" prefixed for symbols in XDEFs and XREFs. They correspond to XREF.B or XDEF.B with the same symbols without the prefix.
Ignore FF character at line start Symbol prefixes (-Compat=f)	With this suboption, an otherwise improper character recognized from feed character is ignored.

Table 3.44 Tool Settings — HCS08 Assembler > Language > Compatibility mode Options

Option	Description
Alternate comment rules (-Compat=c)	With this suboption, comments implicitly start when a space is present after the argument list. A special character is not necessary. Be careful with spaces when this option is given because part of the intended arguments may be taken as a comment. However, to avoid accidental comments, the Assembler does issue a warning if such a comment does not start with a "*" or a ";".
Support FOR directive (-Compat=b)	With this suboption, the Assembler supports a FOR - Repeat assembly block assembly directive to generate repeated patterns more easily without having to use recursive macros.
Add some additional directives (-Compat=a)	With this suboption, some additional directives are added for enhanced compatibility. The Assembler actually supports a SECT directive as an alias of the usual SECTION - Declare Relocatable Section assembly directive. The SECT directive takes the section name as its first argument.
Operator != means equal (-Compat==)	The Assembler takes the default value of the != operator as not equal, as it is in the C language. For compatibility, this behavior can be changed to equal with this option. Because of the risks involved with this option for existing code, a message is issued for every != which is treated as equal.

Table 3.44 Tool Settings — HCS08 Assembler > Language > Compatibility mode Options

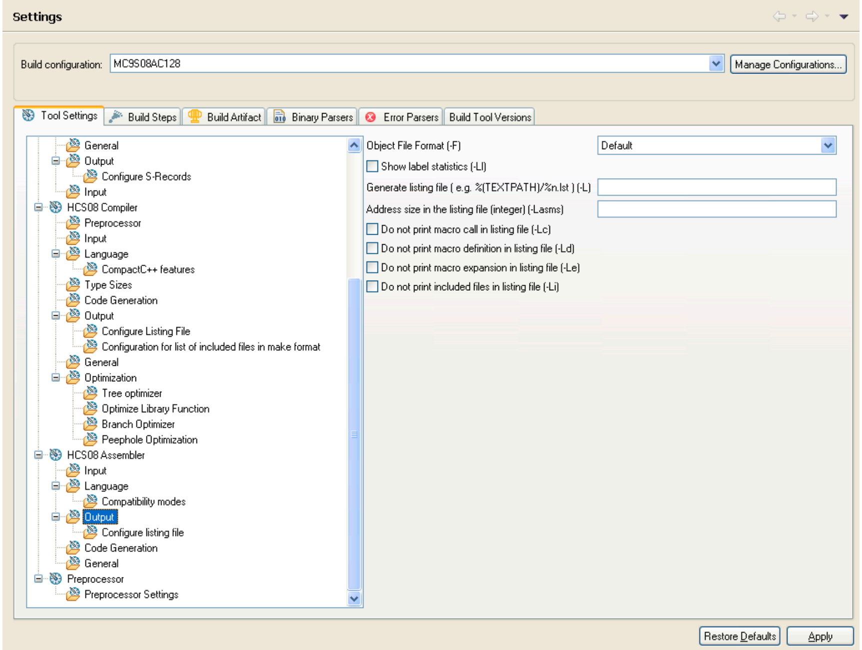
Option	Description
Support $ character in symbol (-Compat=)	With this suboption, the Assembler supports to start identifiers with a $ sign.
Support additional ! symbols (-Compat=!)	The following additional operators are defined when this option is used: • !^: exponentiation • !m: modulo • !@: signed greater or equal • !g: signed greater • !%: signed less or equal • !t: signed less than • !$: unsigned greater or equal • !S: unsigned greater • !&: unsigned less or equal • !l: unsigned less • !n: one complement • !w: low operator • !h: high operator **Note:** The default values for the following ! operators are defined: • !.: binary AND • !x: exclusive OR • !+: binary OR

HCS08 Assembler > Output

Use this panel to control how the assembler generates the output file, as well as error and warning messages. You can specify whether to allocate constant objects in ROM, generate debugging information, and strip file path information.

Figure 3.49 shows the **Output** panel.

Figure 3.49 Tool Settings — HCS08 Assembler > Output


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

Object File Format (-F) Default

☐ Show label statistics (-LI)

Generate listing file (e.g. %(TEXTPATH)/%n.lst) (-L)

Address size in the listing file (integer) (-Lasms)

☐ Do not print macro call in listing file (-Lc)

☐ Do not print macro definition in listing file (-Ld)

☐ Do not print macro expansion in listing file (-Le)

☐ Do not print included files in listing file (-Li)

Table 3.45 lists and describes the output options for HCS08 Assembler.

Table 3.45 Tool Settings — HCS08 Assembler > Output Options

Option	Description
Object File Format (-F)	Defines the format for the output file generated by the Assembler.
Show label statistics (-LI)	Enables the Compiler to append statistical information about the compilation session to the specified file. The information includes Compiler options, code size (in bytes), stack usage (in bytes) and compilation time (in seconds) for each procedure of the compiled file. The Compiler appends the information to the specified filename (or the file make.txt, if no argument given). Set the TEXTPATH: Text File Path environment variable to store the file into the path specified by the environment variable. Otherwise the Compiler stores the file in the current directory.

Table 3.45 Tool Settings — HCS08 Assembler > Output Options (*continued*)

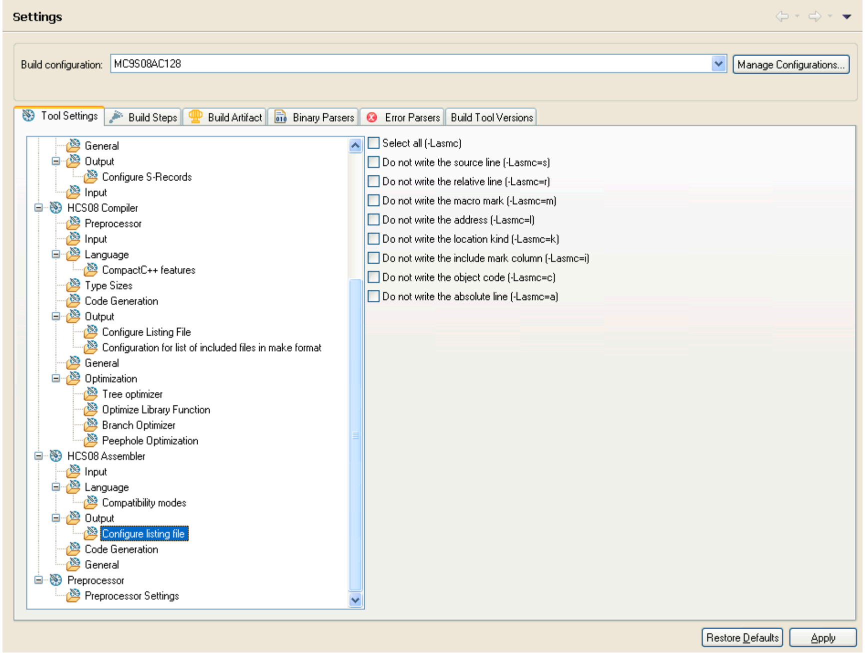
Option	Description
Generate listing file (for example, %(TEXTPATH)/%n.lst) (-L)	Specifies the name, %n, of the assembly listing file. The file is placed in the directory specified by %TEXTPATH. If this option is left blank, no listing file is output.
Address size in the listing file (integer) (-Lasms)	Specifies the size of the addresses displayed in the listing. Options are: • 1 to display addresses as xx • 2 to display addresses as xxxx • 3 to display addresses as xxxxxx • 4 to display addresses asf xxxxxxxx
Do not print macro call in listing file (-Lc)	Specifies whether macro calls encountered in the source code are expanded and appear in the listing file.
Do not print macro definition in listing file (-Ld)	Instructs the Assembler to generate a listing file but not including any macro definitions. The listing file contains macro invocation and expansion lines as well as expanded include files.
Do not print macro expansion in listing file (-Le)	Switches on the generation of the listing file, but macro expansions are not present in the listing file. The listing file contains macro definition and invocation lines as well as expanded include files.
Do not print included files in listing file (-Li)	Switches on the generation of the listing file, but include files are not expanded in the listing file. The listing file contains macro definition, invocation, and expansion lines.

HCS08 Assembler > Output > Configure listing file

Use this panel to specify the general assembler behavior.

Figure 3.50 shows the **Configure listing file** panel.

Figure 3.50 Tool Settings — HCS08 Assembler > Output > Configure listing file


Settings
Build configuration: MC9S08AC128
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

☐ Select all (-Lasmc)
☐ Do not write the source line (-Lasmc=s)
☐ Do not write the relative line (-Lasmc=r)
☐ Do not write the macro mark (-Lasmc=m)
☐ Do not write the address (-Lasmc=l)
☐ Do not write the location kind (-Lasmc=k)
☐ Do not write the include mark column (-Lasmc=i)
☐ Do not write the object code (-Lasmc=c)
☐ Do not write the absolute line (-Lasmc=a)

Table 3.48 lists and describes the configure listing file options for HCS08.

Table 3.46 Tool Settings — Assembler > Output > Configure listing file Options

Option	Description
Select all (-Lasmc)	Print all the columns in the listing file
Do not write the source line (-Lasmc=s)	Do not print source column in the listing file
Do not write the relative line (-Lasmc=r)	Do not print relative column (Rel.) in the listing file
Do not write the macro line (-Lasmc=m)	Do not print macro mark column in the listing file
Do not write the address (-Lasmc=l)	Do not print address column (Loc) in the listing file

Table 3.46 Tool Settings — Assembler > Output > Configure listing file Options

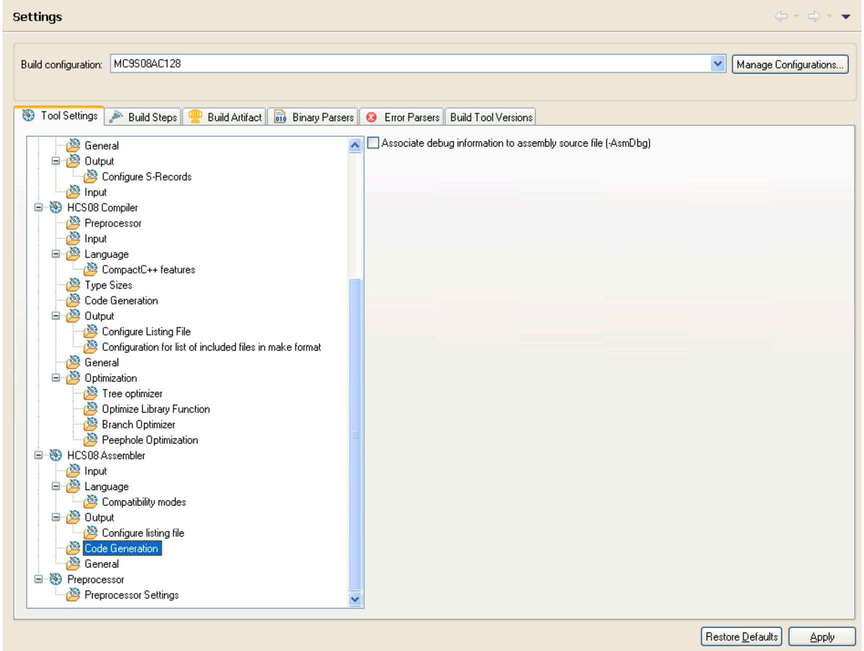
Option	Description
Do not write the location kind (-Lasmc=k)	Do not print the location type column in the listing file
Do not write the include mark column (-Lasmc=i)	Do not print the include mark column in the listing file
Do not write the object code (-Lasmc=c)	Do not print the object code in the listing file
Do not write the absolute line (-Lasmc=a)	Do not print the absolute column (Abs.) in the listing file

HCS08 Assembler > Code Generation

Use this panel to specify the code generation assembler behavior.

Figure 3.52 shows the **Code Generation** panel.

Figure 3.51 Tool Settings — HCS08 Assembler > Code Generation



Build configuration: MC9S08AC128

☐ Associate debug information to assembly source file (-AsmDbg)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.47 lists and describes the code generation assembler options for HCS08.

Table 3.47 Tool Settings — Assembler > Code Generation Options

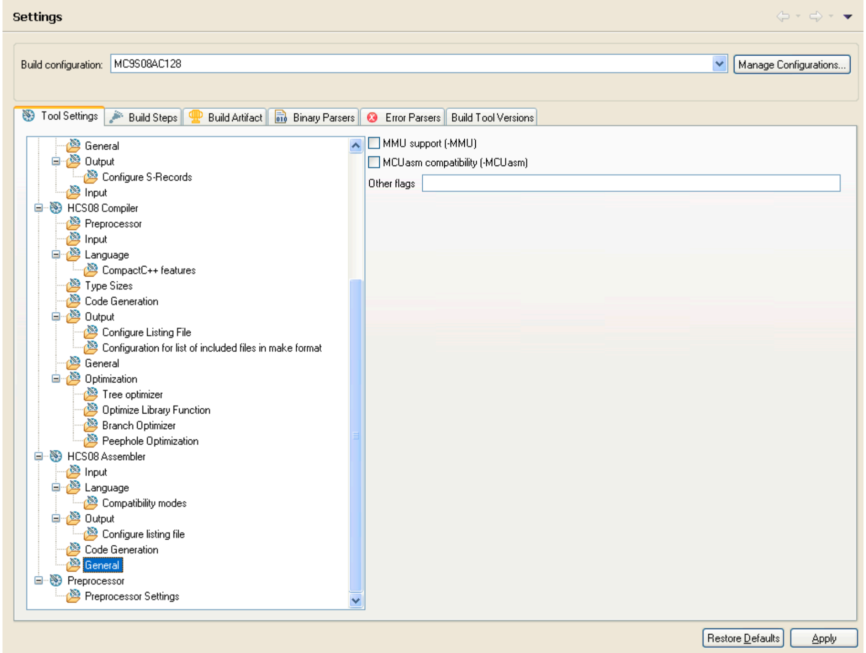
Option	Description
Associate debug information to assembly source file (-AsmDbg)	Passes the assembly source file name information to DWARF sections. When the output .abs file is debugged, the actual assembly source file is displayed instead of intermediary <filename>.dbg file.

HCS08 Assembler > General

Use this panel to specify the general assembler behavior.

Figure 3.52 shows the **General** panel.

Figure 3.52 Tool Settings — HCS08 Assembler > General


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ MMU support (-MMU)

☐ MCUasm compatibility (-MCUasm)

Other flags

Table 3.48 lists and describes the general assembler options for HCS08.

Table 3.48 Tool Settings — Assembler > General Options

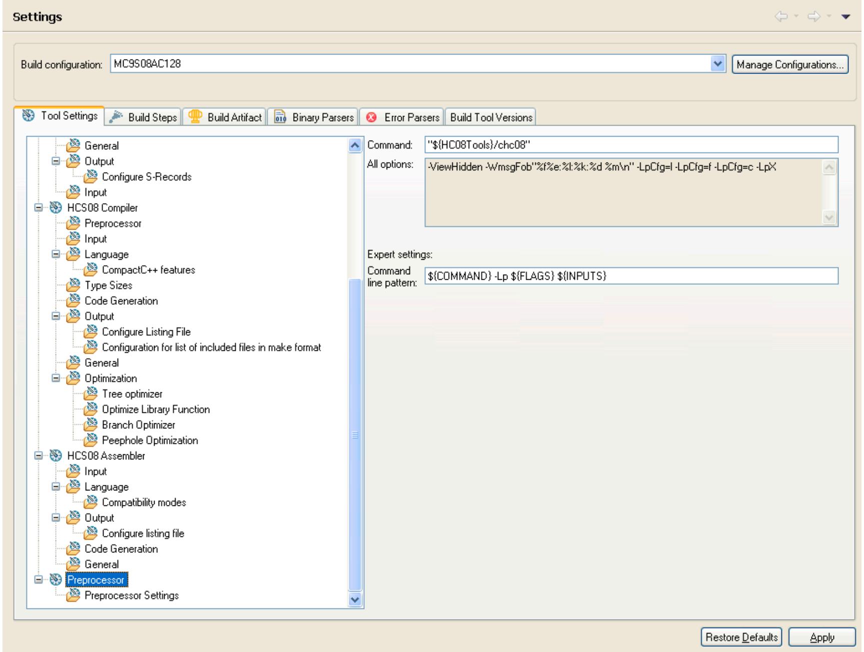
Option	Description
MMU Support (-MMU)	Check to inform the compiler that CALL and RTC instructions are available, enabling code banking, and that the current architecture has extended data access capabilities, enabling support for `__linear` data types. This option can be used only when `-Cs08` is enabled.
MCUasm compatibility (-MCUasm)	Check to activate the compatibility mode with the MCUasm Assembler.
Other Flags	Specify additional command line options for the assembler; type in custom flags that are not otherwise available in the UI.

Preprocessor

Use this panel to specify the preprocessor settings of the HCS08.

Figure 3.53 shows the **Preprocessor** settings.

Figure 3.53 Tool Settings — Preprocessor



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

Command: "${HC08Tools}/chc08"

All options: -ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -LpCfg=l -LpCfg=f -LpCfg=c -LpX

Expert settings:

Command line pattern: ${COMMAND} -Lp ${FLAGS} ${INPUTS}

Table 3.49 lists and describes the preprocessor options for HCS08.

Table 3.49 Tool Settings — Preprocessor Options

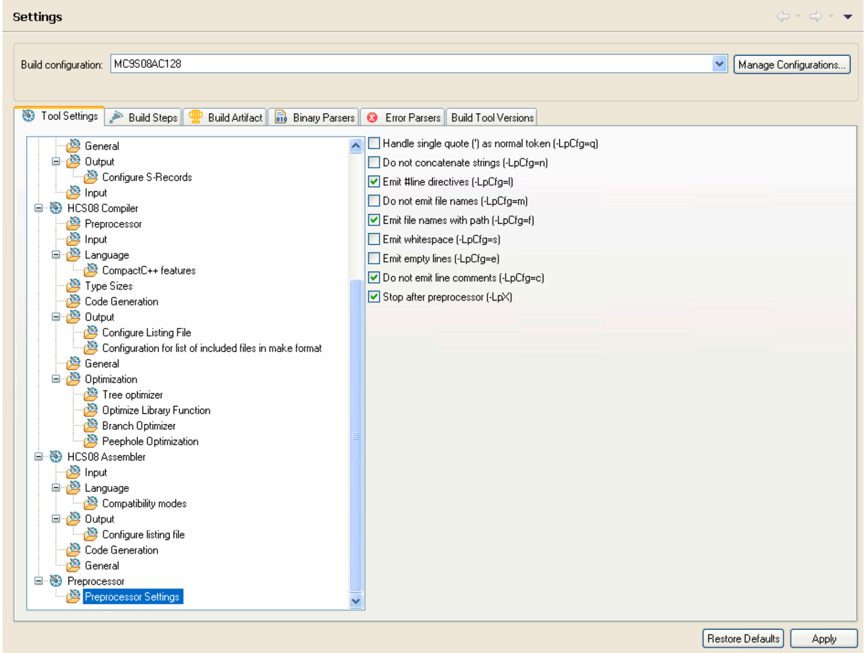
Option	Description
Command	Shows the location of the assembler executable file.
All options	Shows the actual command line the assembler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} -Lp ${FLAGS} ${INPUTS}`.

Preprocessor > Preprocessor Settings

Use this panel to specify the preprocessor settings of HCS08.

Figure 3.53 shows the **Preprocessor > Preprocessor** settings.

Figure 3.54 Tool Settings — Preprocessor > Preprocessor Settings


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
General
Output
Configure S-Records
Input
HCS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Tree optimizer
Optimize Library Function
Branch Optimizer
Peephole Optimization
HCS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9S08AC128

☐ Handle single quote (') as normal token (-LpCfg=q)
☐ Do not concatenate strings (-LpCfg=n)
☑ Emit #line directives (-LpCfg=l)
☐ Do not emit file names (-LpCfg=m)
☑ Emit file names with path (-LpCfg=f)
☐ Emit whitespace (-LpCfg=s)
☐ Emit empty lines (-LpCfg=e)
☑ Do not emit line comments (-LpCfg=c)
☑ Stop after preprocessor (-LpX)

Table 3.50 lists and describes the preprocessor settings options for HCS08.

Table 3.50 Tool Settings — Preprocessor > Preprocessor Settings Options

Option	Description
Handle single quote (‘) as normal token (-LpCfg=q)	Use this option to handle single quote (‘) as normal token.
Do not concatenate strings (-LpCfg=n)	Use this option to avoid string concatenation.
Emit #line directive (-LpCfg=l)	Use this option to emit #line directives in preprocessor output.
Do not emit file names (-LpCfg=m)	Do not emit file names.

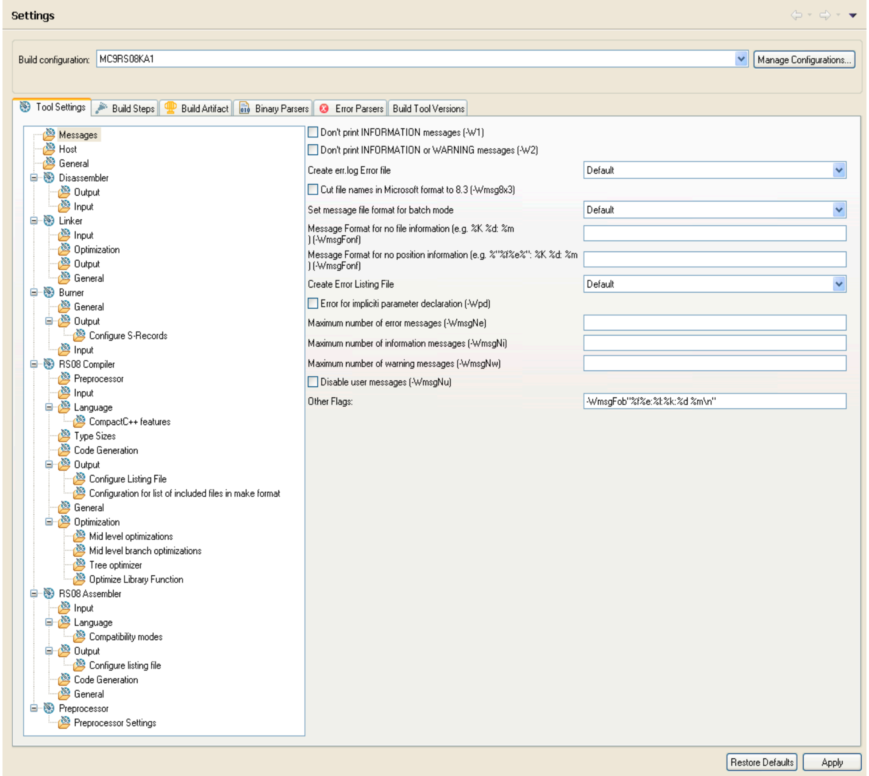
Table 3.50 Tool Settings — Preprocessor > Preprocessor Settings Options (*continued*)

Option	Description
Emit file names with path (-LpCfg=f)	Use this option to emit file names with path.
Emit whitespaces (-LpCfg=s)	Use this option to reconstruct spaces.
Emit empty lines (-LpCfg=e)	Use this option to emit empty lines.
Do not emit line comments (-LpCfg=c)	Do not emit line comments
Stop after preprocessor (-LpX)	Without this option, the compiler always translates the preprocessor output as C code. To do only preprocessing, use this option together with the -Lp option. No object file is generated.

Build Properties for RS08

The **Properties for *<project>*** window shows the corresponding build properties for an RS08 project (Figure 3.55).

Figure 3.55 Build Properties - RS08


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Don't print INFORMATION messages (-W1)

☐ Don't print INFORMATION or WARNING messages (-W2)

Create err.log Error file: Default

☐ Cut file names in Microsoft format to 8.3 (-Wmsg8x3)

Set message file format for batch mode: Default

Message Format for no file information (e.g. %K %d: %m) (-WmsgFonf)

Message Format for no position information (e.g. %"%f%e%": %K %d: %m) (-WmsgFonf)

Create Error Listing File: Default

☐ Error for impliciti parameter declaration (-Wpd)

Maximum number of error messages (-WmsgNe)

Maximum number of information messages (-WmsgNi)

Maximum number of warning messages (-WmsgNw)

☐ Disable user messages (-WmsgNu)

Other Flags: -WmsgFob"%f%e:%l:%k:%d %m\n"

Table 3.51 lists the build properties specific to developing software for HCS08.

The properties that you specify in these panels apply to the selected build tool on the **Tool Settings** page of the **Properties for** ***<project>*** window.

Table 3.51 Build Properties for RS08

Build Tool	Build Properties Panels
Messages	Messages
Host	Host
General	General
Disassembler	Disassembler > Output
	Disassembler > Input

Table 3.51 Build Properties for RS08 (*continued*)

Build Tool	Build Properties Panels
Linker	Linker > Input
	Linker > Optimization
	Linker > Output
	Linker > General
Burner	Burner > General
	Burner > Output • Burner > Output > Configure S-Record
	Burner > Input
RS08 Compiler	RS08 Compiler > Preprocessor
	RS08 Compiler > Input
	RS08 Compiler > Language • RS08 Compiler > Language > CompactC++ features
	RS08 Compiler > Type Sizes
	RS08 Compiler > Code Generation
	RS08 Compiler > Output • HCS08 Compiler > Output > Configure Listing File • HCS08 Compiler > Output > Configuration for list of included files in make format
	RS08 Compiler > General
	RS08 Compiler > Optimization • RS08 Compiler > Optimization > Mid level optimizations • RS08 Compiler > Optimization > Mid level branch optimizations • RS08 Compiler > Optimization > Tree optimizer • RS08 Compiler > Optimization > Optimize Library Function

Table 3.51 Build Properties for RS08 (*continued*)

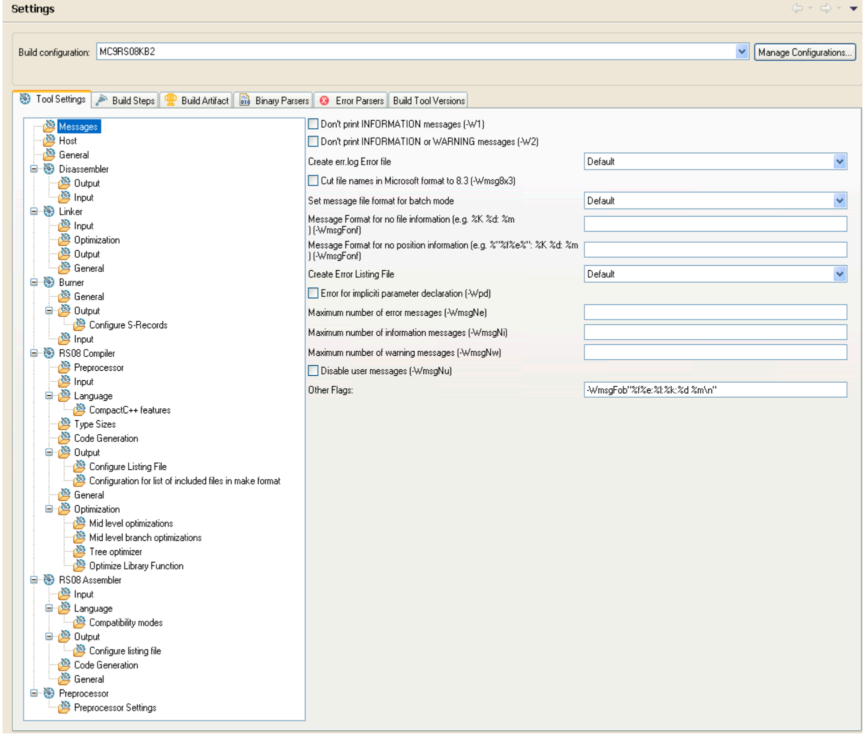
Build Tool	Build Properties Panels
RS08 Assembler	RS08 Assembler > Input
	RS08 Assembler > Language • RS08 Assembler > Language > Compatibility modes
	RS08 Assembler > Output • RS08 Assembler > Output > Configure Listing File
	RS08 Assembler > General
Preprocessor	Preprocessor > Preprocessor Settings

Messages

Use this panel to specify whether to generate symbolic information for debugging the build target

Figure 3.56 shows the **Messages** panel.

Figure 3.56 Tool Settings — Messages


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings

Build configuration: MC9RS08KB2

☐ Don't print INFORMATION messages (-W1)

☐ Don't print INFORMATION or WARNING messages (-W2)

Create err.log Error file: Default

☐ Cut file names in Microsoft format to 8.3 (-Wmsg8x3)

Set message file format for batch mode: Default

Message Format for no file information (e.g. %K %d: %m) (-WmsgFonf):

Message Format for no position information (e.g. %"%f%e%": %K %d: %m) (-WmsgFonf):

Create Error Listing File: Default

☐ Error for impliciti parameter declaration (-Wpd)

Maximum number of error messages (-WmsgNe):

Maximum number of information messages (-WmsgNi):

Maximum number of warning messages (-WmsgNw):

☐ Disable user messages (-WmsgNu)

Other Flags: -WmsgFob"%f%e:%l:%k:%d %m\n"

Table 3.52 lists and and describes the message options.

Table 3.52 Tool Settings — Messages Options

Option	Description
Don’t print INFORMATION messages (-W1)	Inhibits information message reporting. Only warning and error messages are generated.
Don’t print INFORMATION or WARNING messages (-W2)	Suppresses all messages of type INFORMATION and WARNING. Only ERROR messages are generated.

Table 3.52 Tool Settings — Messages Options (*continued*)

Option	Description
Create err.log Error file	Using this option, the Compiler uses a return code to report errors back to the tools. When errors occur, 16-bit window environments use err.log files, containing a list of error numbers, to report the errors. If no errors occur, the 16-bit window environments delete the err.log file.
Cut file names to Microsoft format to 8.3 (-Wmsg8x3)	Some editors (early versions of WinEdit) expect the filename in Microsoft message format (8.3 format). That means the filename can have up to eight characters and no more than a three-character extension. Longer filenames are possible when you use Win95 or WinNT. This option truncates the filename to the 8.3 format.
Set message file format for batch mode	Use this option to start the Compiler with additional arguments (for example, files and Compiler options). If you start the Compiler with arguments (for example, from the Make Tool or with the '%f' argument from the CodeWright IDE), the Compiler compiles the files in a batch mode. No Compiler window is visible and the Compiler terminates after job completion.
Message Format for no file information (e.g. %K %d: %m)(-WmsgFonf)	If there is no file information available for a message, then <string> defines the message format string to use.
Message Format for no positioning information (%"%f%e%":%K %d: %m)(-WmsgFonf)	If there is no position information available for a message, then <string> defines the message format string to use.
Create Error Listing File	This option controls whether the Compiler creates an error listing file. The error listing file contains a list of all messages and errors that occur during processing.

Table 3.52 Tool Settings — Messages Options (*continued*)

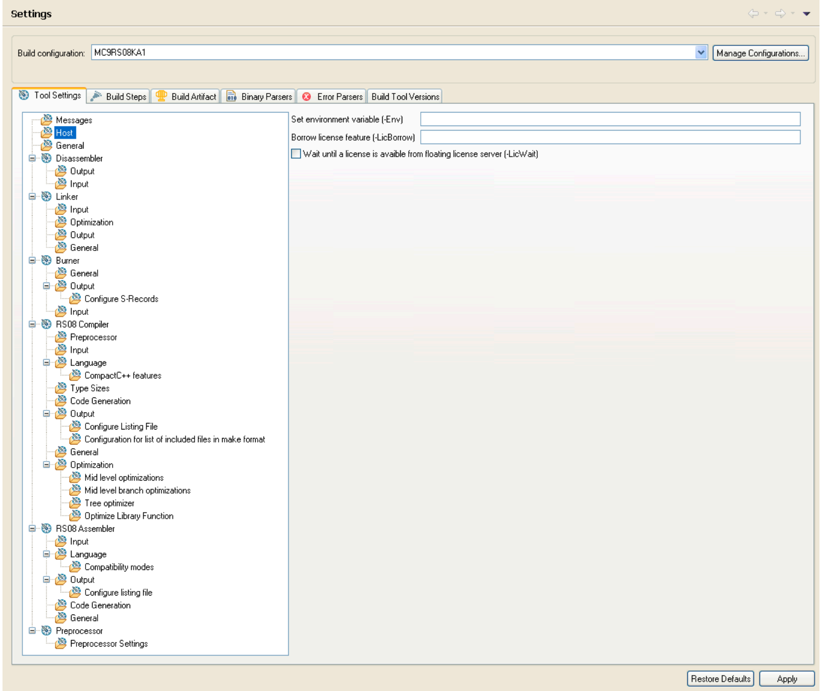
Option	Description
Error for Impliciti parameter declaration (-Wpd)	This option prompts the Compiler to issue an ERROR message instead of a WARNING message when the Compiler encounters an implicit declaration. This occurs if the Compiler does not have a prototype for the called function. This option helps prevent parameter-passing errors, which can only be detected at runtime. It requires prototyping each called function before use. Correct ANSI behavior assumes that parameters are correct for the stated call. This option is the same as using -WmsgSe1801.
Maximum number of error messages (-WmsgNe)	Specify the number of errors allowed until the application stops processing.
Maximum number of information messages (-WmsgNi)	Specify the maximum number of information messages allowed.
Maximum number of warning messages (-WmsgNw)	Specify the maximum number of warnings allowed.
Disable user messages (-WmsgNu)	Check to disable user messages and allow only the normal message categories (WARNING, INFORMATION, ERROR, or FATAL); reduces the number of messages, and simplifies the error parsing of other tools.
Other Flags	Specify additional command line options; type in custom flags that are not otherwise available in the UI. Default value is `-WmsgFob"%%f%%e:%%l: %%k: %%m\n"`

Host

Use this panel to specify the host options of RS08.

Figure 3.57 shows the **Host** panel.

Figure 3.57 Tool Settings — Host



Build configuration: MC9RS08KA1

Set environment variable (-Env)

Borrow license feature (-LicBorrow)

☐ Wait until a license is avaible from floating license server (-LicWait)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.53 lists and describes the memory model options for RS08.

Table 3.53 Tool Settings — General

Option	Description
Set environment variable (-Env)	This option sets an environment variable. Use this environment variable in the maker, or use to overwrite system environment variables.

Table 3.53 Tool Settings — General (*continued*)

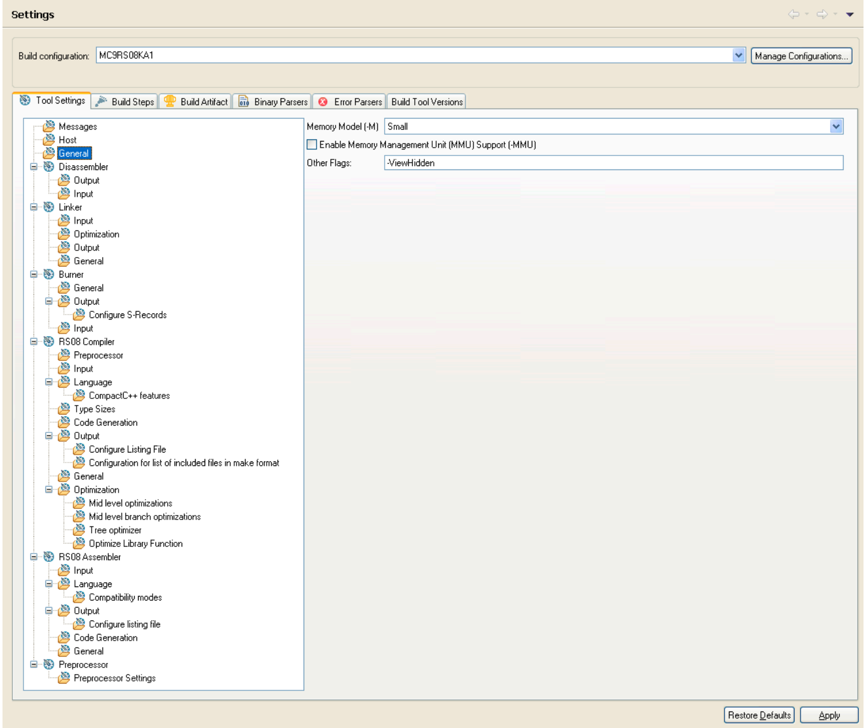
Option	Description
Borrow license feature (-LicBorrow)	This option allows you to borrow a license feature until a given date or time. Borrowing allows you to use a floating license even if disconnected from the floating license server.
Wait until a license is available from floating license server (-LicWait)	By default, if a license is not available from the floating license server, then the application will immediately return. With -LicWait set, the application will wait (blocking) until a license is available from the floating license server.

General

Use this panel to specify the memory model that the architecture uses. The build tools (compiler, linker, and assembler) use the properties that you specify.

Figure 3.58 shows the **General** panel.

Figure 3.58 Tool Settings — General


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Memory Model (-M): Small

☐ Enable Memory Management Unit (MMU) Support (-MMU)

Other Flags: -ViewHidden

Table 3.54 lists and describes the memory model options for RS08.

Table 3.54 Tool Settings — General

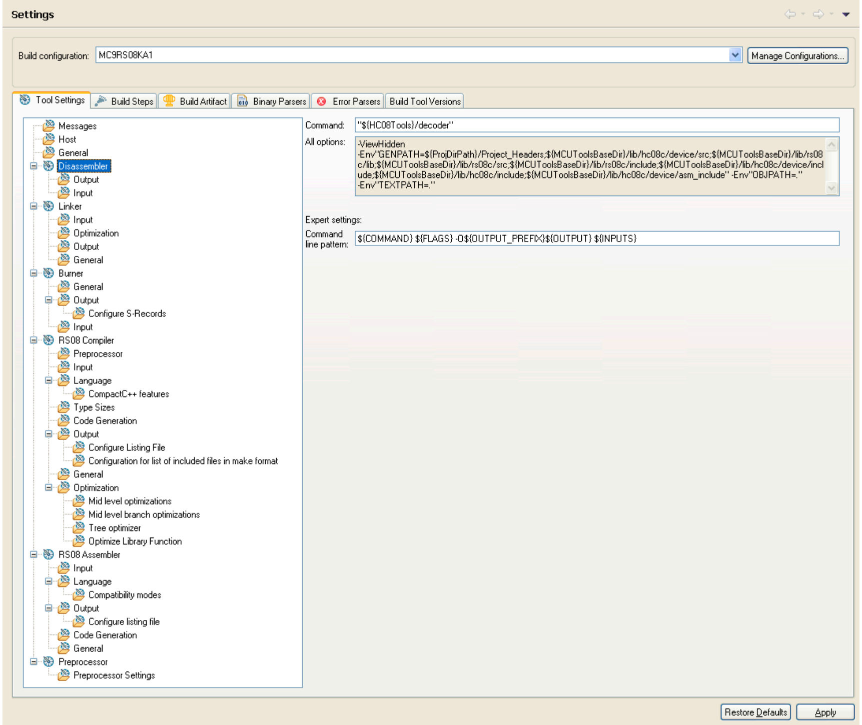
Option	Description
Memory Model (-M)	Specify the memory model for the build tools: • **Tiny** — Assumes that data pointers have 8-bit addresses unless explicitly specified with the keyword __far • **Small** — Default memory model; assumes that all functions and pointers have 16 bit addresses and requires code and data to be located in 64 kilobytes address space • **Banked** — Lets you place program code into atmost 256 pages of 16 kilobytes each, but does not affect data allocation
Enable Memory Management Unit (MMU) Support (-MMU)	Check to inform the compiler that `CALL` and `RTC` instructions are available, enabling code banking, and that the current architecture has extended data access capabilities, enabling support for `__linear` data types. This option can be used only when `-Cs08` is enabled.
Other Flags	Specify additional command line options for the HCS08; type in custom flags that are not otherwise available in the UI.

Disassembler

Use this panel to specify the command, options, and expert settings for RS08 Disassembler.

Figure 3.59 shows the Disassembler page.

Figure 3.59 Tool Settings > Disassembler


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Command: "${HC08Tools}/decoder"

All options:

```
-ViewHidden
-Env"GENPATH=${ProjDirPath}/Project_Headers;${MCUToolsBaseDir}/lib/hc08c/device/src;${MCUToolsBaseDir}/lib/rs08
c/lib;${MCUToolsBaseDir}/lib/rs08c/src;${MCUToolsBaseDir}/lib/rs08c/include;${MCUToolsBaseDir}/lib/hc08c/device/incl
ude;${MCUToolsBaseDir}/lib/hc08c/include;${MCUToolsBaseDir}/lib/hc08c/device/asm_include" -Env"OBJPATH=."
-Env"TEXTPATH=."
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} -O${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.55 lists and describes the Disassembler options.

Table 3.55 Tool Settings — Disassembler Options

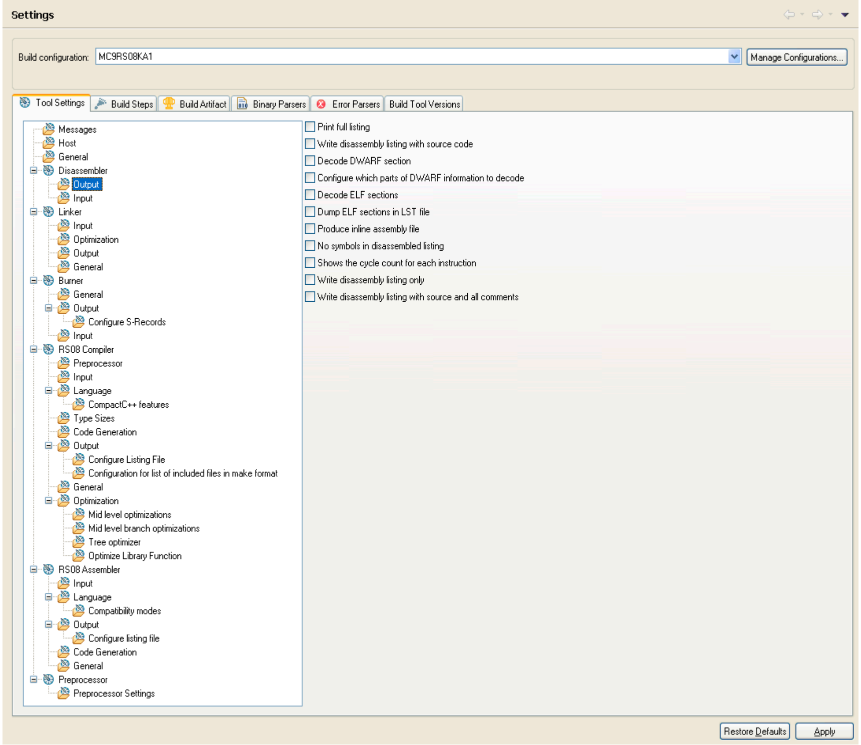
Option	Description
Command	Shows the location of the disassembler executable file; default is `${HC08Tools}/decoder`.
All options	Shows the actual command line the linker will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} -O${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

Disassembler > Output

Use this panel to control how the disassembler generates the output file.

Figure 3.60 shows the **Output** panel.

Figure 3.60 Tool Settings — Disassembler > Output



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Print full listing
☐ Write disassembly listing with source code
☐ Decode DWARF section
☐ Configure which parts of DWARF information to decode
☐ Decode ELF sections
☐ Dump ELF sections in LST file
☐ Produce inline assembly file
☐ No symbols in disassembled listing
☐ Shows the cycle count for each instruction
☐ Write disassembly listing only
☐ Write disassembly listing with source and all comments

Table 3.56 lists and describes the output options for RS08 disassembler.

Table 3.56 Tool Settings — Disassembler > Output Options

Option	Description
Print full listing	Prints a listing with the header information of the object file.
Write disassembly listing with source code	Check to enable the decoder decoding Freescale object files write the source code within the disassembly listing. This option setting is default for the Freescale object files as input.
Decode DWARF section	Check to write the DWARF section information in the listing file. Decoding from the DWARF section inserts this information in the listing file.
Configure which parts of DWARF in formation to decode	Check to configure parts of DWARF in formation to decode.
Decode ELF sections	Check to ensure that the ELF section information is also written to the listing file. Decoding from the ELF section inserts the following information in the listing file.
Dump ELF sections	Check to generate a HEX dump of all ELF sections.
Dump ELF sections in LST file	Check to generate a HEX dump of all ELF sections in a LST file.
Produce inline assembly file	Check to ensure that the output listing is an inline assembly file without additional information, but in C comments.
No symbols in disassembled listing	Check to prevent symbols from printing in the disassembled listing.
Shows the cycle count for each instruction	Check to ensure that each instruction line contains the count of cycles in '[',']' braces. The cycle count is written before the mnemonics of the instruction. Note that the cycle count display is not supported for all architectures.

Table 3.56 Tool Settings — Disassembler > Output Options (*continued*)

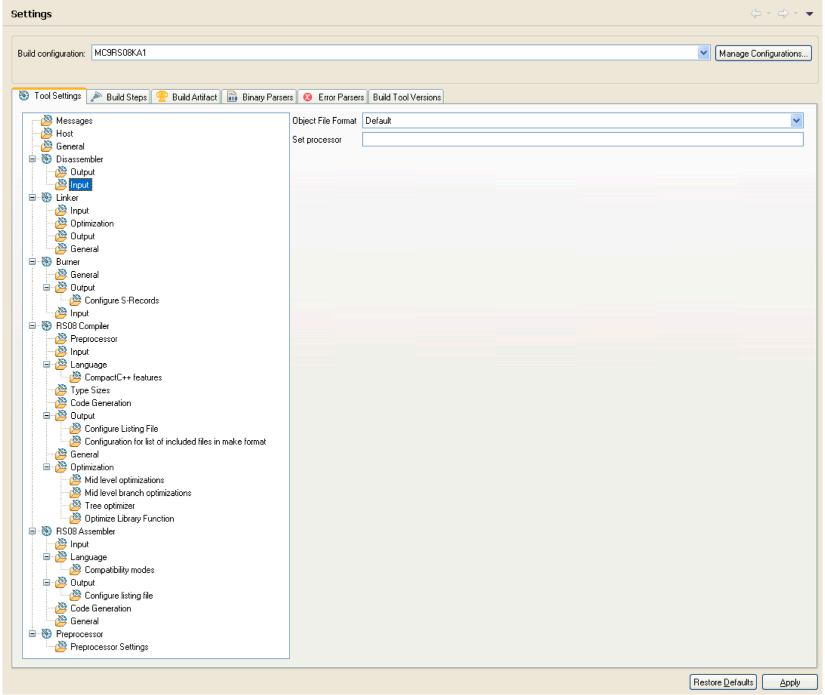
Option	Description
Write disassembly listing only	Check to ensure that the Decoder decoding Freescale object files writes the source code within the disassembly listing only.
Write disassembly listing with source and all comments	Check to write the origin source and its comments within the disassembly listing.

Disassembler > Input

Use this panel to control how the disassembler generates the input file.

Figure 3.61 shows the **Input** panel.

Figure 3.61 Tool Settings — Disassembler > Input



Build configuration: MC9RS08KA1

Object File Format: Default

Set processor:

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.57 lists and describes the Disassembler > Input options for RS08.

Table 3.57 Tool Settings — Disassembler > Input Options

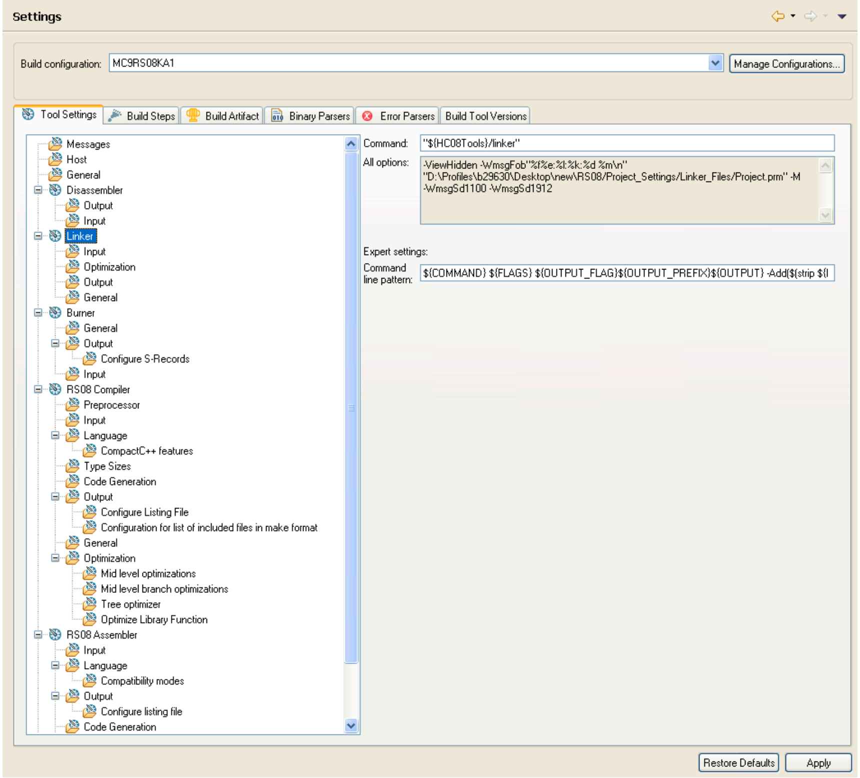
Option	Description
Object File Format	Defines the format for the output file generated by the disassembler.
Set Processor	

Linker

Use this panel to specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the general, libraries, and search path settings.

Figure 3.62 shows the **Linker** settings.

Figure 3.62 Tool Settings — Linker


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Command: "${HC08Tools}/linker"

All options:

```
-ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n"
"D:\Profiles\b29630\Desktop\new\RS08/Project_Settings/Linker_Files/Project.prm" -M
-WmsgSd1100 -WmsgSd1912
```

Expert settings:

Command line pattern: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} -Add($(strip ${I`

Table 3.58 lists and describes the linker options for RS08.

Table 3.58 Tool Settings — Linker Options

Option	Description
Command	Shows the location of the linker executable file.
All options	Shows the actual command line the linker will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS}${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} -add( ${INPUTS} ).`

Linker > Input

Use this panel to specify the parameter file path, startup function, object file search paths, and any additional libraries that the **C/C++ Linker** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

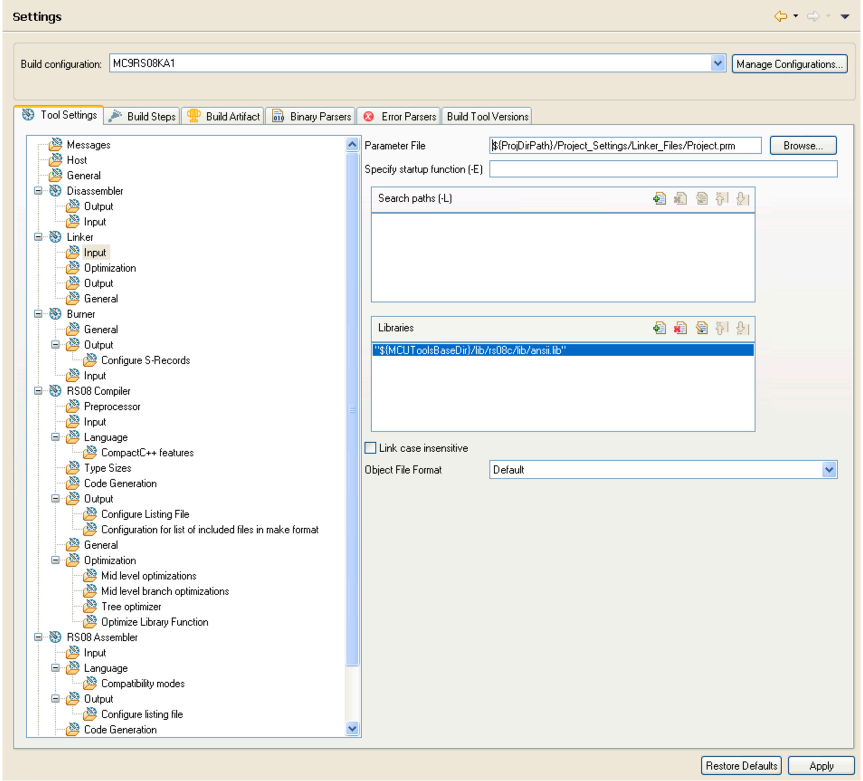
The IDE first looks for an include file in the current directory, or the directory that you specify in the `INCLUDE` directive. If the IDE does not find the file, it continues searching the paths shown in this panel. The IDE keeps searching paths until it finds the `#include` file or finishes searching the last path at the bottom of the Include File Search Paths list. The IDE appends to each path the string that you specify in the `INCLUDE` directive.

NOTE The IDE displays an error message if a header file is in a different directory from the referencing source file. Sometimes, the IDE also displays an error message if a header file is in the same directory as the referencing source file.

For example, if you see the message `Could not open source file myfile.h`, you must add the path for `myfile.h` to this panel.

Figure 3.63 shows the **Input** panel.

Figure 3.63 Tool Settings — Linker > Input


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Browse...
Search paths (-L)
Libraries
Restore Defaults
Apply
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation

Build configuration: MC9RS08KA1

Parameter File: ${ProjDirPath}/Project_Settings/Linker_Files/Project.prm

Specify startup function (-E):

Libraries: "${MCUToolsBaseDir}/lib/rs08c/lib/ansii.lib"

☐ Link case insensitive

Object File Format: Default

Table 3.59 lists and describes the linker input options for RS08.

Table 3.59 Tool Settings — Linker > Input Options

Option	Description
Parameter File	Shows the path of the parameter file. Default value is: `${ProjDirPath}/Project_Settings/Linker_Files/Project.prm`.
Specify startup function (-E)	Tells the command-line tool to preprocess source files.

Table 3.59 Tool Settings — Linker > Input Options (*continued*)

Option	Description
Search paths (-L)	Shows the list of all search paths; the ELF part of the linker searches object files first in all paths and then the usual environment variables are considered.
Libraries	Lists paths to additional libraries that the C/C++ linker uses
Link case insensitive	
Object File Format	Defines the format for the output file generated by the Linker.

Table 3.60 lists and describes the toolbar buttons that help work with the libraries and the additional object file search paths.

Table 3.60 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.64) and specify the object file search path.
✖	**Delete** — Click to delete the selected object file search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.65) and update the selected object file search path.
⇧	**Move up** — Click to move the selected object file search path one position higher in the list.
⇩	**Move down** — Click to move the selected object file search path one position lower in the list.

Figure 3.64 shows the **Add directory path** dialog box.

Figure 3.64 Add directory path Dialog Box

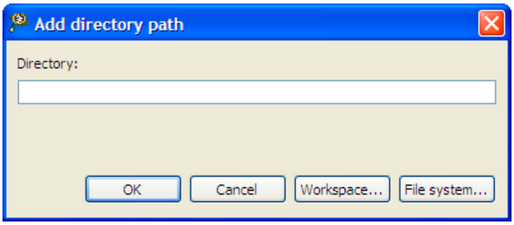
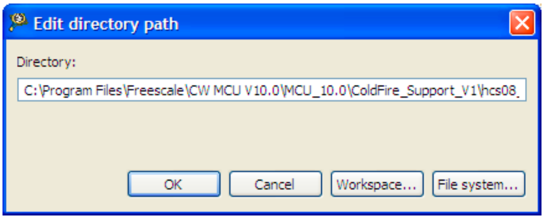


Figure 3.65 shows the **Edit directory path** dialog box.

Figure 3.65 Edit directory path Dialog Box



Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

Table 3.61 lists and describes the toolbar buttons that help work with the libraries and the additional object files.

Table 3.61 Libraries Toolbar Buttons

Button	Description
+	**Add** — Click to open the **Add file path** dialog box (Figure 3.66) and specify location of the library you want to add.
✖	**Delete** — Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.

Table 3.61 Libraries Toolbar Buttons (*continued*)

Button	Description
✎	**Edit —** Click to open the **Edit file path** dialog box (Figure 3.67) and update the selected path.
⇧	**Move up —** Click to move the selected path one position higher in the list.
⇩	**Move down —** Click to move the selected path one position lower in the list.

Figure 3.66 shows the **Add file path** dialog box.

Figure 3.66 Tool Settings — Linker > Libraries - Add file path Dialog Box

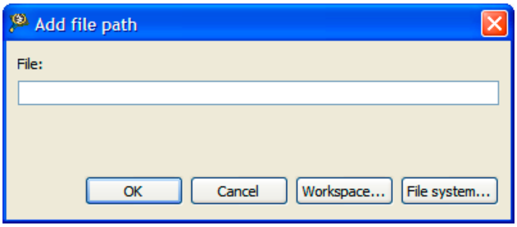
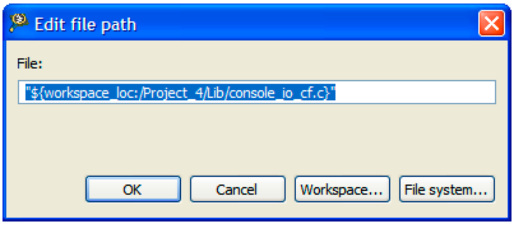


Figure 3.67 shows the **Edit file path** dialog box.

Figure 3.67 Tool Settings — Linker > Libraries - Edit file path Dialog Box



"${workspace_loc:/Project_4/Lib/console_io_cf.c}"

The buttons in the **Add file path and Edit file path** dialog boxes help work with the file paths.

- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **File Selection** dialog box and specify the file path. The resulting path, relative to the workspace, appears in the appropriate list.

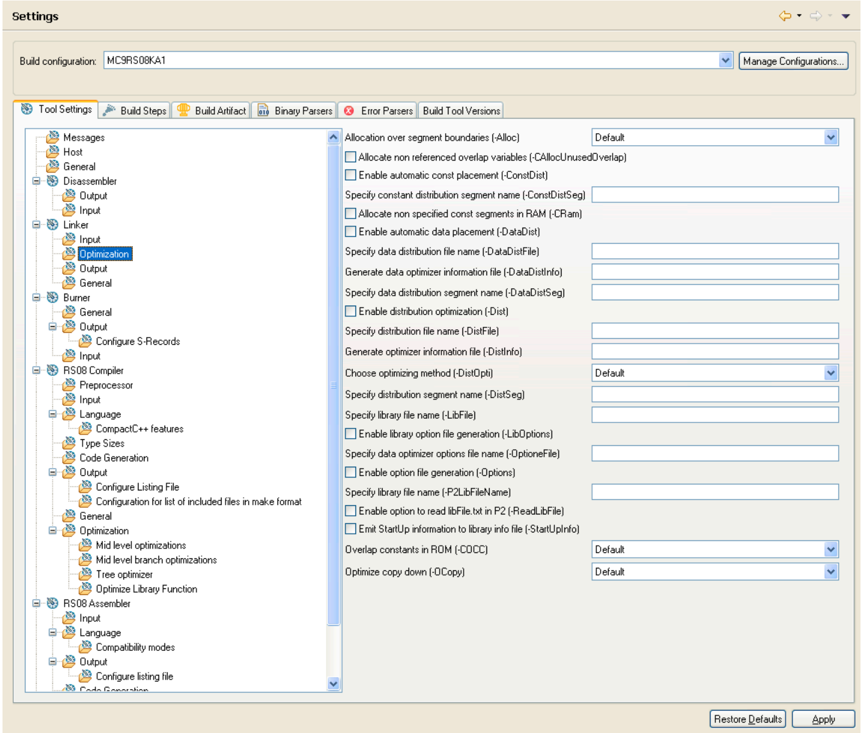
- **File system —** Click to display the **Open** dialog box and specify the file path. The resulting absolute path appears in the appropriate list.

Linker > Optimization

Use this panel to control linker optimizations. The linker's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.68 shows the **Optimization** panel.

Figure 3.68 Tool Settings — Linker > Optimizations


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Allocation over segment boundaries (-Alloc) Default
☐ Allocate non referenced overlap variables (-CAllocUnusedOverlap)
☐ Enable automatic const placement (-ConstDist)
Specify constant distribution segment name (-ConstDistSeg)
☐ Allocate non specified const segments in RAM (-CRam)
☐ Enable automatic data placement (-DataDist)
Specify data distribution file name (-DataDistFile)
Generate data optimizer information file (-DataDistInfo)
Specify data distribution segment name (-DataDistSeg)
☐ Enable distribution optimization (-Dist)
Specify distribution file name (-DistFile)
Generate optimizer information file (-DistInfo)
Choose optimizing method (-DistOpti) Default
Specify distribution segment name (-DistSeg)
Specify library file name (-LibFile)
☐ Enable library option file generation (-LibOptions)
Specify data optimizer options file name (-OptioneFile)
☐ Enable option file generation (-Options)
Specify library file name (-P2LibFileName)
☐ Enable option to read libFile.txt in P2 (-ReadLibFile)
☐ Emit StartUp information to library info file (-StartUpInfo)
Overlap constants in ROM (-COCC) Default
Optimize copy down (-OCopy) Default

Table 3.62 lists and describes the linker optimization options for RS08.

Table 3.62 Tool Settings — Linker > Optimization Options

Option	Description
Allocation over segment boundaries (-Alloc)	The linker supports to allocate objects from one ELF section into different segments. The allocation strategy controls where space for the next object is allocated as soon as the first segment is full. Options are: • **Always use next segment:** In the AllocNext strategy, the linker always takes the next segment as soon as the current segment is full. Holes generated during this process are not used later. With this strategy, the allocation order corresponds to the definition order in the object files. Objects defined first in a source file are allocated before later defined objects. • **Always check for free previous segment:** In the AllocFirst strategy, the linker checks for every object, if there is a previously only partially used segment, into which the current object does fit. This strategy does not maintain the definition order. • **Check for free previous segment when current is full:** In the AllocChange strategy, the linker checks as soon as a object does no longer fit into the current segment, if there is a previously only partially used segment, into which the current object does fit. This strategy does not maintain the definition order, but it does however use fewer different ranges than the AllocFirst case.
Allocate non referenced overlap variables (-CAllocUnusedOverlap)	When Smart Linking is switched off, defined but unreferenced overlapped variables are not allocated by default. Such variables do not belong to a specific function, therefore they cannot be allocated overlapped with other variables. This option only changes the behavior of variables in the special _OVERLAP segment. This segment is used only to allocate parameters and local variables for processors which do not have a stack. Not allocating an unreferenced overlap variable is similar to not allocating a variable on the stack for other processors. If you use this stack analogy, then allocating such variables this way corresponds to allocating unreferenced stack variables in global memory. This option allows allocation of all defined objects. Using this option is not recommended.

Table 3.62 Tool Settings — Linker > Optimization Options

Option	Description
Enable automatic const placement (-ConstDist)	With this option the linker constant optimizer is enabled. Instead of performing usual linking actions, the linker generates a data distribution file which contains optimized distribution for constant objects.
Specify constant distribution segment name (-ConstDistSeg)	When this option is enabled, it's possible to specify the name of the constant distribution segment.
Allocate non specified const segments in RAM (-CRam)	This option allocates constant data segments not explicitly allocated in a READ_ONLY segment in the default READ_WRITE segment. This was the default for old versions of the linker, so this option provides a compatible behavior with old linker versions.
Enable automatic data placement (-DataDist)	With this option the linker data optimizer is enabled. Instead of performing usual linking actions, the linker generates a data distribution file which contains optimized distribution.
Specify data distribution name (-DataDistFile)	When this option is enabled, it's possible to specify the name of the data distribution file. There, all distributed data and how the compiler has to reallocate them are listed.
Generate data optimizer information file (-DataDistInfo)	When this option is enabled, the data optimizer generates a data distribution information file giving information on object to segment mapping
Specify data distribution segment name (-DataDistSeg)	When this option is enabled, it's possible to specify the name of the data distribution segment.
Enable distribution optimization (-Dist)	This option enables the linker optimizer. Instead of a link, the linker generates a distribution file which contains an optimized distribution.
Specify distribution file name (-DistFile)	Enable this option to specify the name of the distribution file. The distribution file lists all distributed functions and specifies how the compiler reallocates them.
Generate optimizer information file (-DistInfo)	Using this option, the optimizer generates a distribution information file containing a list of all sections and their functions. Available function information includes the old size, optimized size, and new calling convention.

Table 3.62 Tool Settings — Linker > Optimization Options

Option	Description
Choose optimizing method (-DistOpti)	Enable this option to choose the optimizing method. With the FillBanks argument the linker minimizes the free space in every bank. FillBanks is most effective for functions using the near calling convention. Use the CodeSize argument to minimize code when free space within the banks is no concern.
Specify distribution segment name (-DistSeg)	Use this option to specify the name of the distribution segment.
Specify library file name (-LibFile)	When this option is enabled,linker generates file<filename> which has information about the current libraries and also about the files with which they should be replaced with.
Enable library option file generation (-LibOptions)	Enables library information generation. When this option is enabled,linker generates file (default libFile.txt) which has information about the current library and the startup file and also about the files with which they should be replaced with.
Specify data optimizer options file name (--OptioneFile)	Specifies the name of the file that contains the set of linker-generated compiler options. When this option is enabled, linker places the second step compiler options in the specified file<filename>.
Enable option file generation (-Option)	Enables compiler option generation. The generated options will be used for second step compilation. Linker generates a text file containing a compiler option for the second step (one of the following: -ConstQualiNear, -NonConstQualiNear, -Mb). The content of the file is appended to the compiler options for the second compilation step.
Specify library file name (-P2LibFileName)	Specifies the name of the library information file. When this option is enabled in second link step,linker reads file<filename> which has information about the libraries.
Enable option to read libFile.txt in P2 (-ReadLibFile)	Instructs the linker to read in the library information file that it generated in step one. This option is passed in second link step. It tells the linker to read library information file(default libFile.txt).

Table 3.62 Tool Settings — Linker > Optimization Options

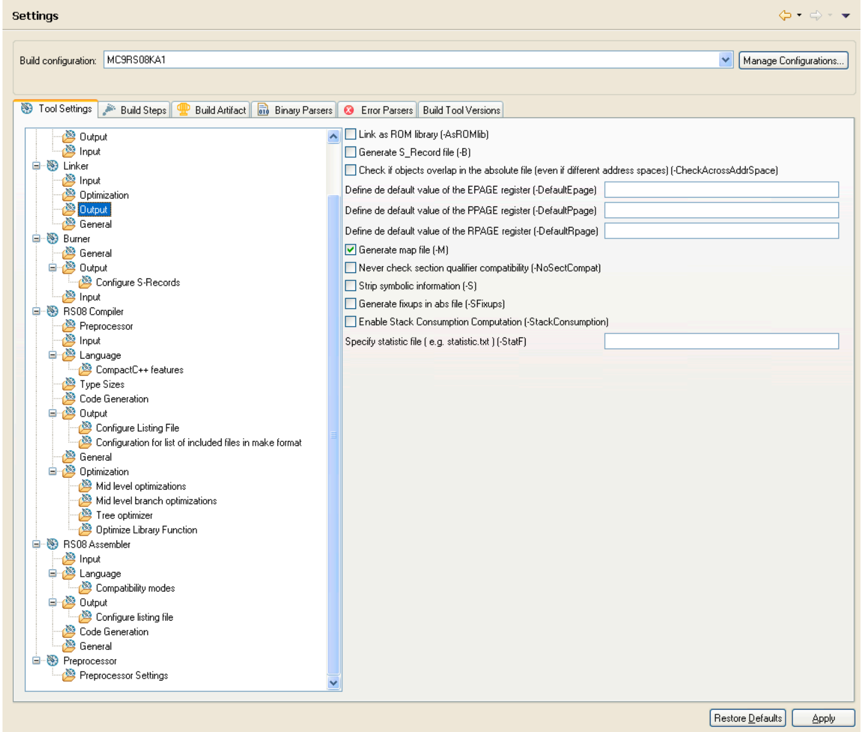
Option	Description
Emit StartUp information to library info file (-StartUpInfo)	The information about the current startup file and the replacement startup file will be added to the library file(default libFile.txt) and used during the second compile-link step.
Overlap constants in ROM (-COCC)	Defines the default if constants and code should be optimized. The commands DO_OVERLAP_CONSTS and DO_NOT_OVERLAP_CONSTS take precedence over the option. Options are: • **Default** • **Overlap constant data and code** • **Overlap constant data** • **Overlap code**
Optimize copy down (-OCopy)	Changes the copy down structure to use few spaces. The optimization does assume that the application does perform both the zero out and the copy down step of the global initialization. If a value is set to zero by the zero out, then zero values are removed from the copy down information. The resulting initialization is not changed by this optimization if the default startup code is used. Options are: • **Enable** • **Disable**

Linker > Output

Use this panel to control how the linker formats the listing file, as well as error and warning messages.

Figure 3.69 shows the **Output** panel.

Figure 3.69 Tool Settings — Linker > Output


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Link as ROM library (-AsROMlib)
☐ Generate S_Record file (-B)
☐ Check if objects overlap in the absolute file (even if different address spaces) (-CheckAcrossAddrSpace)
Define de default value of the EPAGE register (-DefaultEpage)
Define de default value of the PPAGE register (-DefaultPpage)
Define de default value of the RPAGE register (-DefaultRpage)
☑ Generate map file (-M)
☐ Never check section qualifier compatibility (-NoSectCompat)
☐ Strip symbolic information (-S)
☐ Generate fixups in abs file (-SFixups)
☐ Enable Stack Consumption Computation (-StackConsumption)
Specify statistic file (e.g. statistic.txt) (-StatF)

Table 3.63 lists and describes the linker output options for RS08.

Table 3.63 Tool Settings — Linker > Output Options

Option	Description
Link as ROM library (-AsROMlib)	Check to link the application as a ROM library. This option has the same effect as specifying AS ROM_LIB in the linker parameter file.
Generate S_record file (-B)	Check to specify that in addition to an absolute file, also an srecord file should be generated. The name of the srecord file is the same as the name of the abs file, except that the extension SX is used. The default env variable SRECORD may specify an alternative extension.

Table 3.63 Tool Settings — Linker > Output Options (*continued*)

Option	Description
Check if objects overlap in the absolute file (even if different address spaces) (-CheckAcrossAddrSpace)	Check to instruct the linker to check if objects overlap, taking into account their address space.
Define de default value of the EPAGE register (-DefaultEpage)	Defines the reset value for the EEPROM Page Index Register (EPAGE). The value is specific to the actual S12(X) derivative.
Define de default value of the PPAGE register (-DefaultPpage)	Defines the reset value for the Program Page Index Register (PPAGE). The value is specific to the actual S12(X) derivative.
Define de default value of the RPAGE register (-DefaultRpage)	Defines the reset value for the RAM Page Index Register (RPAGE). The value is specific to the actual S12(X) derivative.
Generate map file (-M)	Forces the generation of a map file after a successful linking session.
Never check section qualifier compatibility (-NoSectCompat)	For some target CPU's, when placing a section in a segment the linker checks if the qualifiers of the section are compatible with the ones of the segment (for instance when placing .text into RAM may result in a linker error).This option disables the check.
Strip symbolic information (-S)	Check to disable the generation of DWARF sections in the absolute file to save memory space.
Generate fixups in abs file (-SFixups)	Check to ensure compatibility with previous linker versions. Usually, absolute files do not contain any fixups because all fixups are evaluated at link time. But with fixups, the decoder might symbolically decode the content in absolute files. Some debuggers do not load absolute files which contain fixups because they assume that these fixups are not yet evaluated. But the fixups inserted with this option are actually already handled by this linker.

Table 3.63 Tool Settings — Linker > Output Options (*continued*)

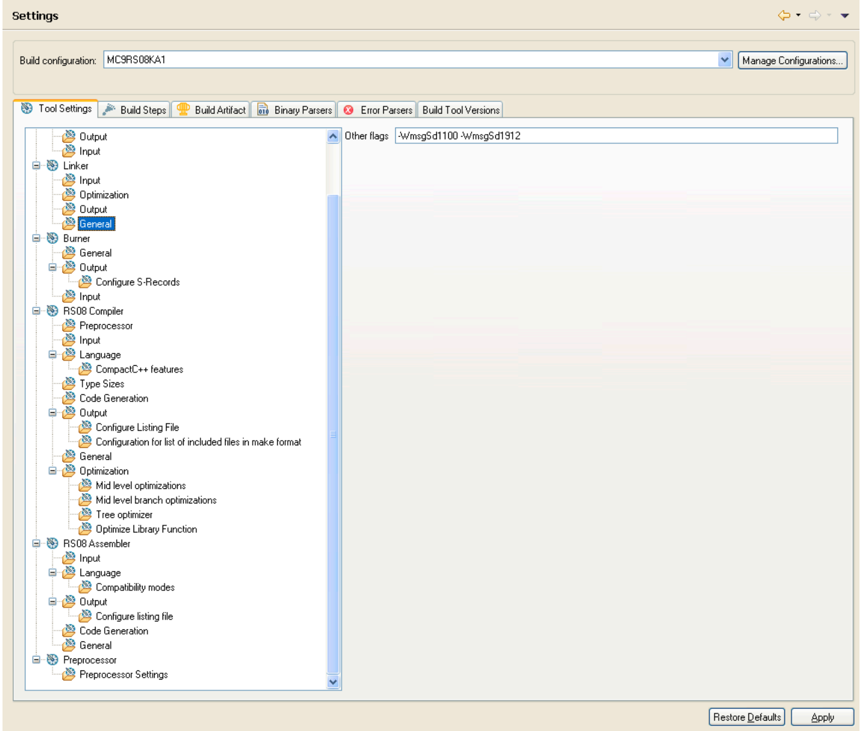
Option	**Description**
Enable Stack Consumption Computation (-StackConsumption)	The linker computes maximum stack effect for given application when the option is enabled and places the result in the output .map file.
Specify statistic file (e.g. statistic.txt) (-StatF)	Specify the name of the linker statistic file. The statistic file reports each allocated object and its attributes. Every attribute is separated by a tab character, so it can be easily imported into a spreadsheet/database program for further processing.

Linker > General

Use this panel to specify the general linker behavior.

Figure 3.70 shows the **General** panel.

Figure 3.70 Tool Settings — Linker > General



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Other flags -WmsgSd1100 -WmsgSd1912

Table 3.64 lists and describes the general linker options for RS08.

Table 3.64 Tool Settings — Linker > General Options

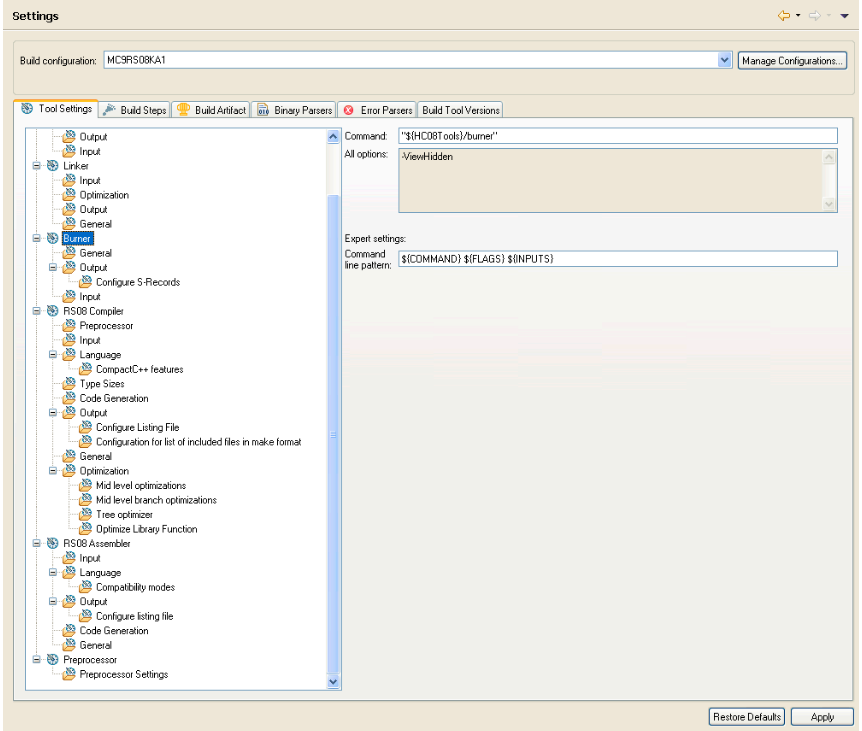
Option	Description
Other flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI. default value is `-WmsgSd1100 -WmsgSd1912`.

Burner

Use the Burner for RS08 Preference Panel to map *.bbl (batch burner language) files to the Burner Plug-In. When the project folder contains a *.bbl file, *.bbl file processing during the post-link phase uses the settings in the Burner preference panel.

Figure 3.71 shows the RS08 Burner settings.

Figure 3.71 Tool Settings > Burner


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Command: "${HC08Tools}/burner"

All options: -ViewHidden

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${INPUTS}

Table 3.65 lists and describes the burner options for RS08.

Table 3.65 Tool Settings — Burner Options

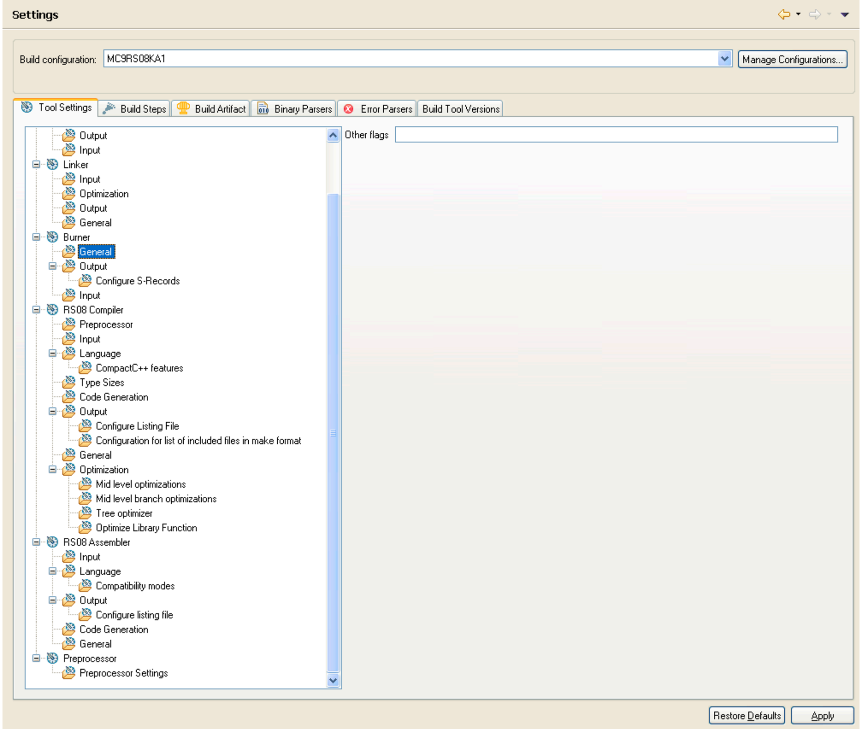
Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${HC08Tools}/burner"`
All options	Shows the actual command line the burner will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${INPUTS}`.

Burner > General

Use this panel to specify other flags for the RS08 Burner to use.

Figure 3.72 shows the **General** panel.

Figure 3.72 Tool Settings — Burner > General



Build configuration: MC9RS08KA1

Other flags:

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.66 lists and describes the general options for RS08 burner.

Table 3.66 Tool Settings — Burner > General Options

Option	Description
Other flags	Specify additional command line options for the burner; type in custom flags that are not otherwise available in the UI.

Burner > Output

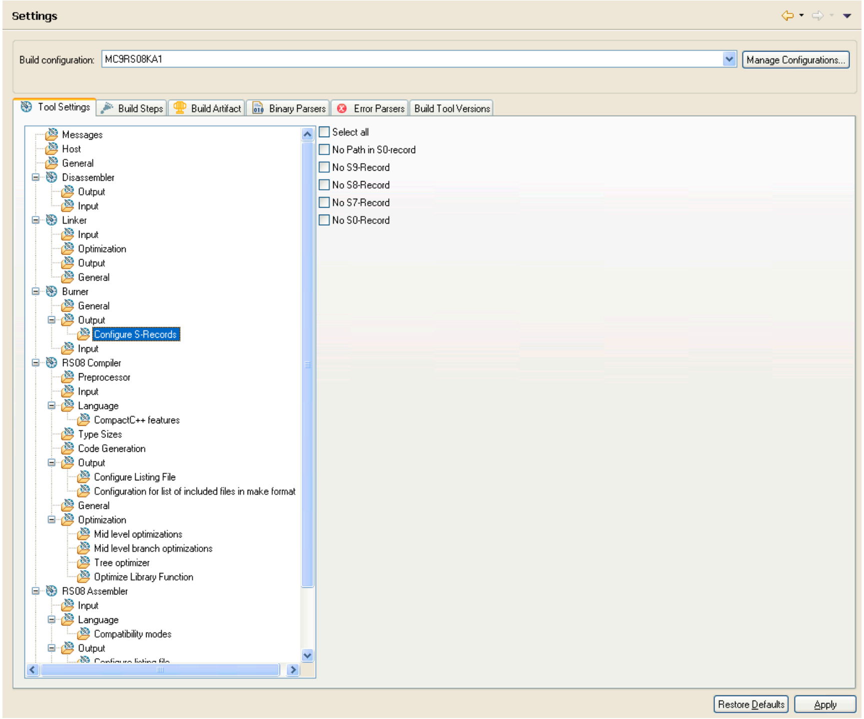
Use this panel to specify the output options for RS08.

Burner > Output > Configure S-Record

Use this panel to configure the S-record options of the Burner.

Figure 3.74 shows the Configure S-Record panel.

Figure 3.73 Tool Settings — RS08 Burner > Output > Configure S-Record



Settings
Build configuration: MC9RS08KA1
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Messages
Host
General
Disassembler
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Restore Defaults
Apply

☐ Select all
☐ No Path in S0-record
☐ No S9-Record
☐ No S8-Record
☐ No S7-Record
☐ No S0-Record

Table 3.66 lists and describes the Configure S-Record options for RS08 burner.

Table 3.67 Tool Settings — Burner > General

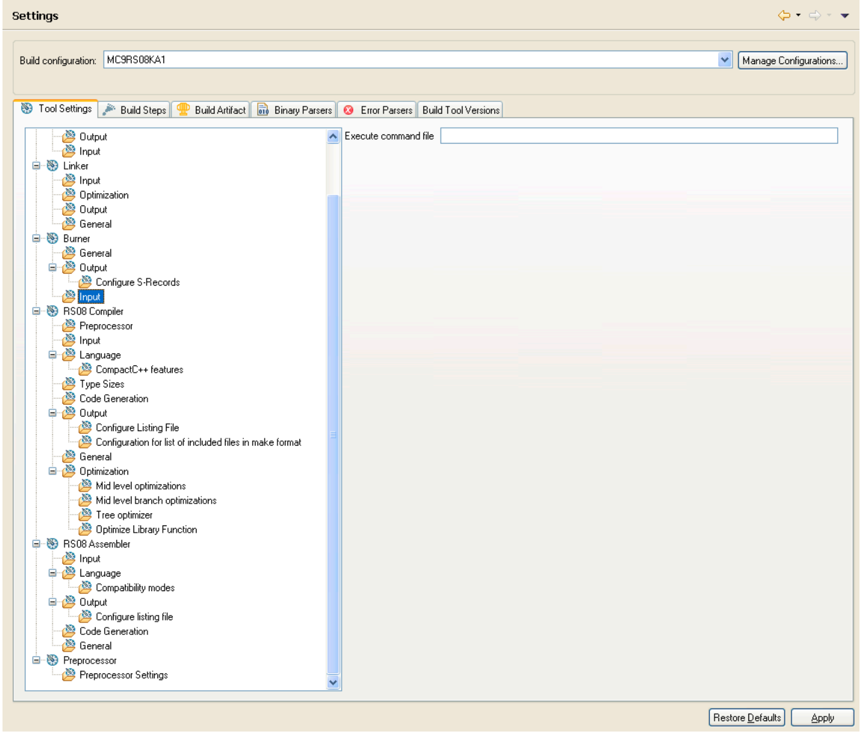
Option	Description
Select all	Disables generation of all start (S0) and end records (S7, S8, or S9)
No Path in S0-record	Removes the path (if present) from the file name in the S0 record
No S9-record	Disables generation of S9-record
No S8-record	Disables generation of S8-record
No S7-record	Disables generation of S7-record
No S0-record	Disables generation of S0-record

Burner > Input

Use this panel to specify the execute command file of the Burner input.

Figure 3.74 shows the Input panel.

Figure 3.74 Tool Settings — Burner > Input



Build configuration: MC9RS08KA1

Execute command file

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.69 lists and describes the input options for burner.

Table 3.68 Tool Settings — Compiler Options

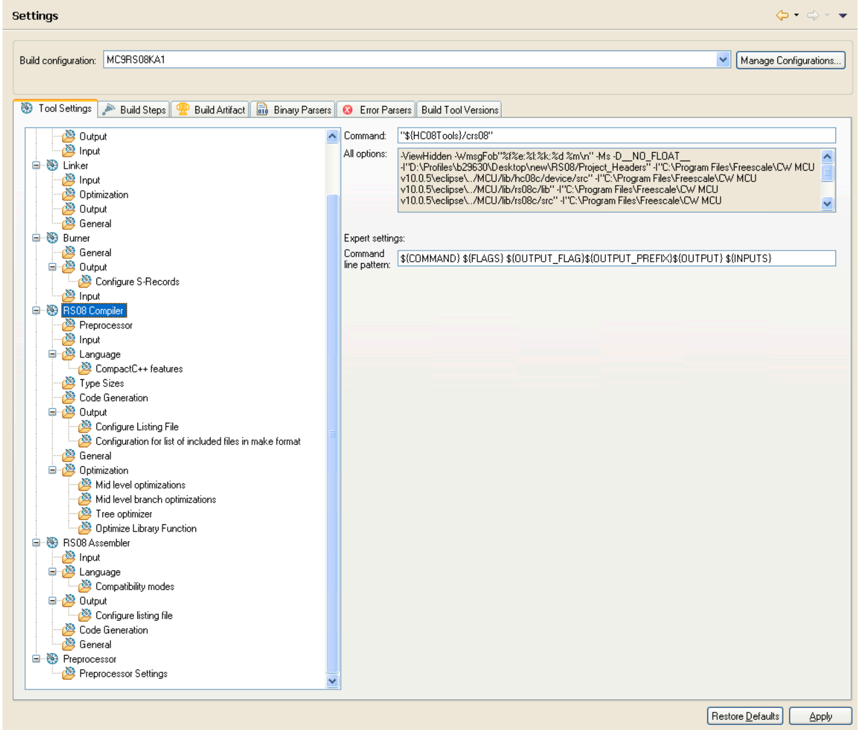
Option	Description
Execute command file	This option causes the Burner to execute a Batch Burner command file (usual extension is .bbl).

RS08 Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the RS08 Compiler tree control includes the general and the file search path settings.

Figure 3.75 shows the RS08 Compiler settings.

Figure 3.75 Tool Settings — RS08 Compiler



Build configuration: MC9RS08KA1

Command: "${HC08Tools}/crs08"

All options:

```
-ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -Ms -D__NO_FLOAT__
-I"D:\Profiles\b29630\Desktop\new\RS08/Project_Headers" -I"C:\Program Files\Freescale\CW MCU
v10.0.5\eclipse\../MCU/lib/hc08c/device/src" -I"C:\Program Files\Freescale\CW MCU
v10.0.5\eclipse\../MCU/lib/rs08c/lib" -I"C:\Program Files\Freescale\CW MCU
v10.0.5\eclipse\../MCU/lib/rs08c/src" -I"C:\Program Files\Freescale\CW MCU
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.69 lists and describes the compiler options for RS08

Table 3.69 Tool Settings — Compiler Options

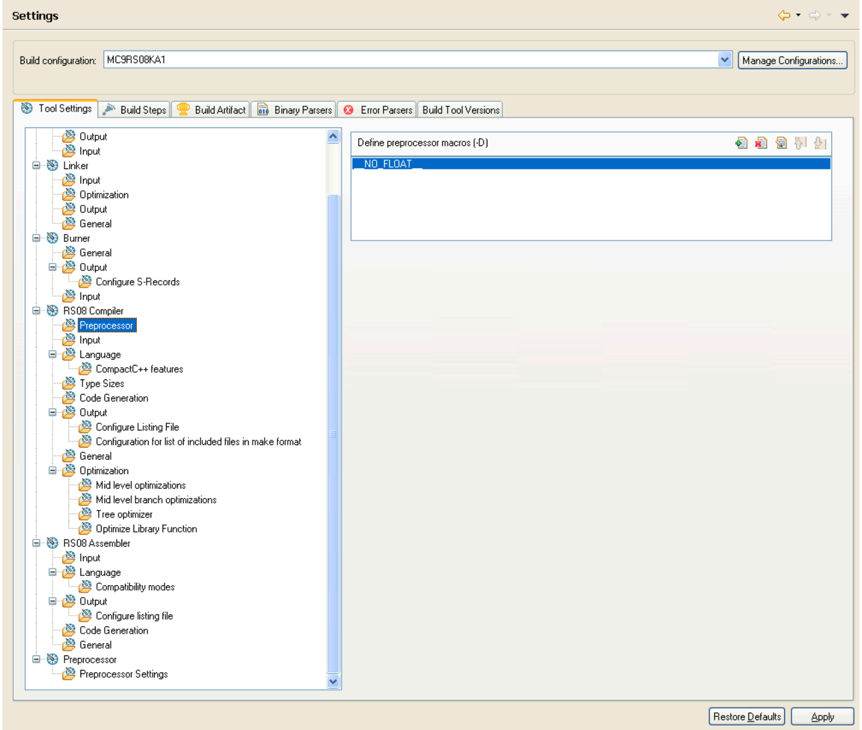
Option	Description
Command	Shows the location of the linker executable file. Default value is : "`${HC08Tools}/crs08.exe`"
All options	Shows the actual command line the compiler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `{COMMAND} ${FLAGS}${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

RS08 Compiler > Preprocessor

Use this panel to specify preprocessor behavior and define macros.

Figure 3.76 shows the **Preprocessor** panel.

Figure 3.76 Tool Settings — RS08 Compiler > Preprocessor



Build configuration: MC9RS08KA1

Define preprocessor macros (-D)

`__NO_FLOAT__`

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.70 lists and describes the preprocessor options for RS08 Compiler.

Table 3.70 Tool Settings — RS08 Compiler > Preprocessor Options

Option	Description
Define preprocessor macros (-D)	Define, delete, or rearrange preprocessor macros. You can specify multiple macros and change the order in which the IDE uses the macros. Define preprocessor macros and optionally assign their values. This setting is equivalent to specifying the `-D` *`name`*`[=`*`value`*`]` command-line option. To assign a value, use the equal sign (=) with no white space. For example, this syntax defines a preprocessor value named `EXTENDED_FEATURE` and assigns `ON` as its value: `EXTENDED_FEATURE=ON` Note that if you do not assign a value to the macro, the shell assigns a default value of 1.

Table 3.71 lists and describes the toolbar buttons that help work with preprocessor macro definitions.

Table 3.71 Define Preprocessor Macros Toolbar Buttons

Button	Description
✚	**Add** — Click to open the **Enter Value** dialog box (Figure 3.77) and specify the path/macro.
✖	**Delete** — Click to delete the selected path/macro. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit Dialog** dialog box (Figure 3.78) and update the selected path/macro.

Table 3.71 Define Preprocessor Macros Toolbar Buttons (*continued*)

Button	Description
⇧	**Move up** — Click to move the selected path/ macro one position higher in the list.
⇩	**Move down** — Click to move the selected path/macro one position lower in the list

Figure 3.77 shows the **Enter Value** dialog box.

Figure 3.77 Tool Settings — RS08 Compiler > Preprocessor - Enter Value Dialog Box

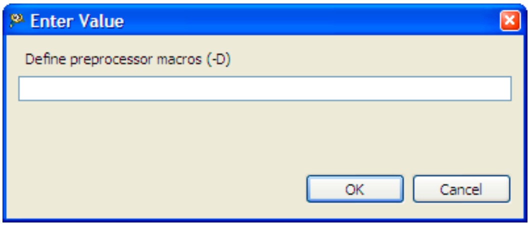
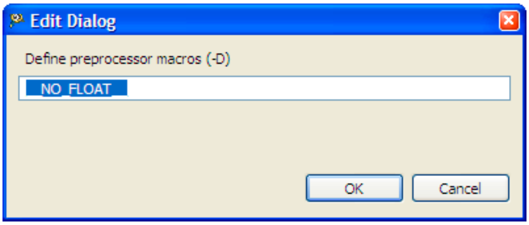


Figure 3.78 shows the **Edit Dialog** dialog box.

Figure 3.78 Tool Settings — RS08 Compiler > Preprocessor - Edit Dialog Box



Define preprocessor macros (-D): NO_FLOAT

The buttons in the **Enter Value and Edit** dialog boxes help work with the preprocessor macros.

- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.

RS08 Compiler > Input

Use this panel to specify file search paths and any additional include files the **RS08 Compiler** should use. You can specify multiple search paths and the order in which you want to perform the search.

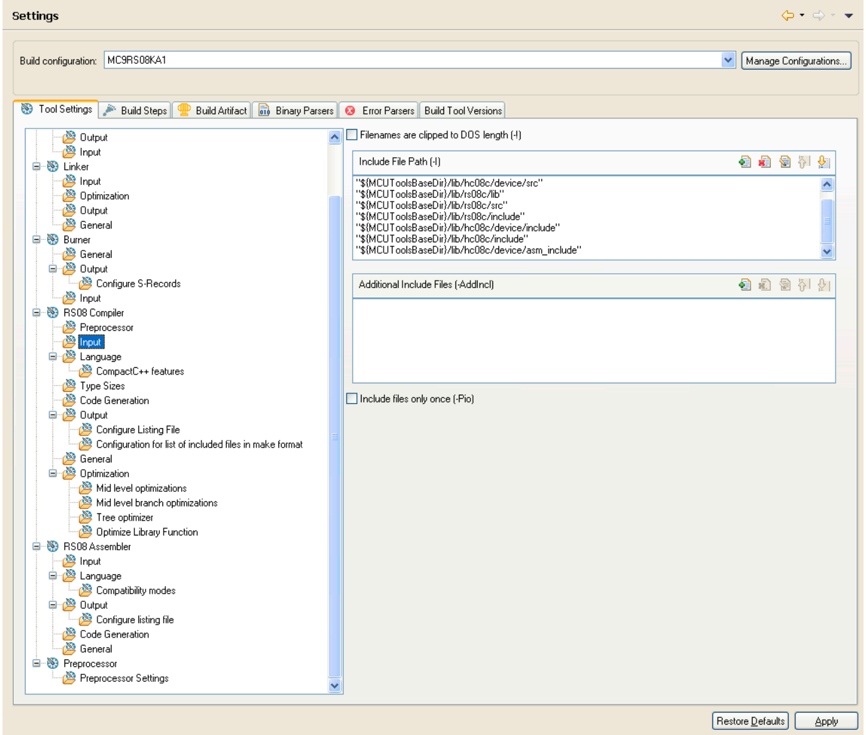
The IDE first looks for an include file in the current directory, or the directory that you specify in the `INCLUDE` directive. If the IDE does not find the file, it continues searching the paths shown in this panel. The IDE keeps searching paths until it finds the `#include` file or finishes searching the last path at the bottom of the Include File Search Paths list. The IDE appends to each path the string that you specify in the `INCLUDE` directive.

NOTE The IDE displays an error message if a header file is in a different directory from the referencing source file. Sometimes, the IDE also displays an error message if a header file is in the same directory as the referencing source file.

For example, if you see the message `Could not open source file myfile.h`, you must add the path for `myfile.h` to this panel.

Figure 3.79 shows the **Input** panel.

Figure 3.79 Tool Settings — RS08 Compiler > Input


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Filenames are clipped to DOS length (-!)

Include File Path (-I)

```
"${MCUToolsBaseDir}/lib/hc08c/device/src"
"${MCUToolsBaseDir}/lib/rs08c/lib"
"${MCUToolsBaseDir}/lib/rs08c/src"
"${MCUToolsBaseDir}/lib/rs08c/include"
"${MCUToolsBaseDir}/lib/hc08c/device/include"
"${MCUToolsBaseDir}/lib/hc08c/include"
"${MCUToolsBaseDir}/lib/hc08c/device/asm_include"
```

Additional Include Files (-AddIncl)

☐ Include files only once (-Pio)

Table 3.72 lists and describes the input options for RS08 Compiler.

Table 3.72 Tool Settings — RS08 Compiler > Input Options

Option	Description
Filenames are clipped to DOS length (-!)	The filenames are clipped to DOS length (eight characters), when compiling files from MS-DOS file system.
Include File Path (-I)	Specify, delete, or rearrange file search paths.
Additional Include Files (-AddInd)	Specify, delete, or rearrange paths to search any additional #include files.
Include files only once	Check to include every header file only once; duplicates are ignored.

Table 3.73 lists and describes the toolbar buttons that help work with the file paths.

Table 3.73 Include File Path (-I) Toolbar Buttons

Button	Description
➕	**Add —** Click to open the **Add directory path** dialog box (Figure 3.80) and specify location of the library you want to add.
✖	**Delete —** Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.81) and update the selected path.
⇧	**Move up —** Click to move the selected path one position higher in the list.
⇩	**Move down —** Click to move the selected path one position lower in the list.

Table 3.74 lists and describes the toolbar buttons that help work with the search paths.

Table 3.74 Additional Include Files (-AddIncl) Toolbar Buttons

Button	Description
➕	**Add —** Click to open the **Add directory path** dialog box (Figure 3.80) and specify location of the library you want to add.
✖	**Delete —** Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.81) and update the selected path.
⇧	**Move up —** Click to move the selected path one position higher in the list.
⇩	**Move down —** Click to move the selected path one position lower in the list.

Figure 3.80 Tool Settings — RS08 Compiler > Input - Add file path Dialog Box

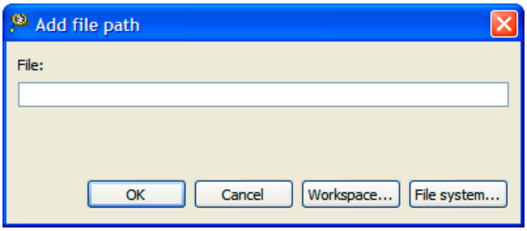
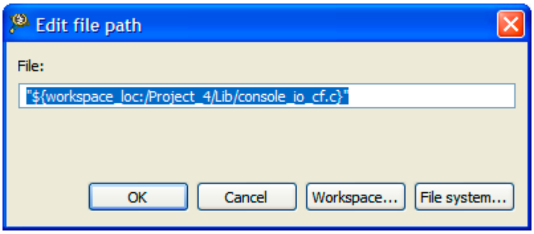


Figure 3.81 Tool Settings — RS08 Compiler > Input - Edit file path Dialog Box



File: "${workspace_loc:/Project_4/Lib/console_io_cf.c}"

The buttons in the **Add file path** (Figure 3.80) **and Edit file path** (Figure 3.81) dialog boxes help work with the paths.

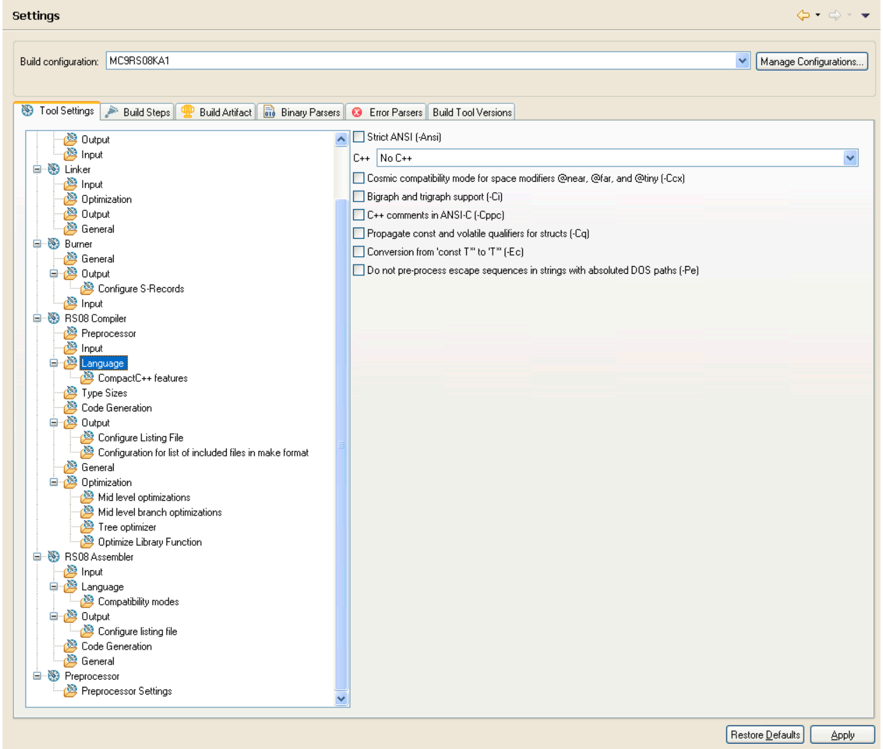
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **File Selection** dialog box and specify the path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Open** dialog box and specify the path. The resulting path appears in the appropriate list.

RS08 Compiler > Language

Use this panel to specify code- and symbol-generation options for the RS08 Compiler.

Figure 3.82 shows the **Language** panel.

Figure 3.82 Tool Settings — RS08 Compiler > Language



Build configuration: MC9RS08KA1

☐ Strict ANSI (-Ansi)

C++ No C++

☐ Cosmic compatibility mode for space modifiers @near, @far, and @tiny (-Ccx)

☐ Bigraph and trigraph support (-Ci)

☐ C++ comments in ANSI-C (-Cppc)

☐ Propagate const and volatile qualifiers for structs (-Cq)

☐ Conversion from 'const T*' to 'T*' (-Ec)

☐ Do not pre-process escape sequences in strings with absoluted DOS paths (-Pe)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.75 lists and describes the language options for RS08.

Table 3.75 Tool Settings — RS08 Compiler > Language Options

Option	Description
Strict ANSI	Check if you want the C compiler to operate in strict ANSI mode. In this mode, the compiler strictly applies the rules of the ANSI/ISO specification to all input files. This setting is equivalent to specifying the `-ansi` command-line option. The compiler issues a warning for each ANSI/ISO extension it finds.
C++	With this option enabled, the Compiler behaves as a C++ Compiler. You can select between three different types of C++: • **Full C++ (-C++f) —** Supports the whole C++ language. • **Embedded C++ (-C++e)** — Supports a constant subset of the C++ language. EC++ does not support inefficient things like templates, multiple inheritance, virtual base classes and exception handling. • **CompactC++ (-C++c) —** Supports a configurable subset of the C++ language. You can configure this subset with the option -Cn. • **No C++** — If the option is not set, the Compiler behaves as an ANSI-C Compiler. If the option is enabled and the source file name extension is *.c, the Compiler behaves as a C++ Compiler. If the option is not set, but the source filename extension is .cpp or .cxx, the Compiler behaves as if the -C++f option were set.

Table 3.75 Tool Settings — RS08 Compiler > Language Options (*continued*)

Option	Description
Cosmic compatibility mode for space modifiers @near, @far, and @tiny (-Ccx)	Check to allow Cosmic style @near, @far and @tiny space modifiers as well as @interrupt in your C code. The -ANSI option must be switched off. It is not necessary to remove the Cosmic space modifiers from your application code. There is no need to place the objects to sections addressable by the Cosmic space modifiers. The following is done when a Cosmic modifier is parsed: The objects declared with the space modifier are always allocated in a special Cosmic compatibility (_CX) section (regardless of which section pragma is set) depending on the space modifier, on the const qualifier or if it is a function or a variable. Space modifiers on the left hand side of a pointer declaration specify the pointer type and pointer size, depending on the target.
Bigraph and trigraph support (-Ci)	Check to replace certain unavailable tokens with the equivalent keywords.
C++ comments in ANSI-C (-Cppc)	Check to allow C++ comments.
Propagate const and colatile qualifiers for structs (-Cq)	Check to propagate const and volatile qualifiers for structures. If all members of a structure are constant or volatile, the structure itself is constant or volatile. If the structure is declared as constant or volatile, all its members are constant or volatile, respectively.

Table 3.75 Tool Settings — RS08 Compiler > Language Options (*continued*)

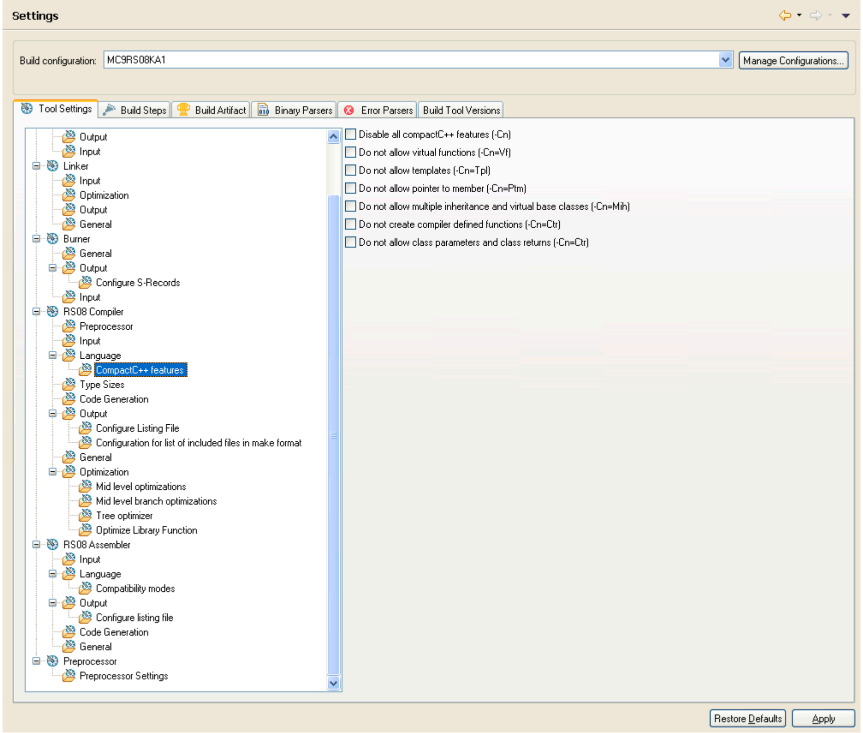
Option	Description
Conversion from ‘const T*’ to ‘T*’ (-Ec)	Check to enable this non-ANSI compliant extension allows the compiler to treat a pointer to a constant type like a pointer to the non-constant equivalent of the type. Earlier Compilers did not check a store to a constant object through a pointer. This option is useful when compiling older source code.
Do not pre-process escape sequences in strings with absoluted DOS paths (-Pe)	If escape sequences are used in macros, they are handled in an include directive similar to the way they are handled in a printf() instruction. If the -Pe option is used, escape sequences are ignored in strings that contain a DOS drive letter ('a – 'z', 'A' – 'Z') followed by a colon ':' and a backslash '\'. When the -Pe option is enabled, the Compiler handles strings in include directives differently from other strings. Escape sequences in include directive strings are not evaluated.

RS08 Compiler > Language > CompactC++ features

Use this panel to select compact C++ features of RS08 compiler.

Figure 3.83 shows the **CompactC++ features** panel.

Figure 3.83 Tool Settings — RS08 Compiler > Language > CompactC++ features


Settings
Build configuration:
MC9RS08KA1
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

☐ Disable all compactC++ features (-Cn)
☐ Do not allow virtual functions (-Cn=Vf)
☐ Do not allow templates (-Cn=Tpl)
☐ Do not allow pointer to member (-Cn=Ptm)
☐ Do not allow multiple inheritance and virtual base classes (-Cn=Mih)
☐ Do not create compiler defined functions (-Cn=Ctr)
☐ Do not allow class parameters and class returns (-Cn=Ctr)

Table 3.76 lists and describes the compactC++ options for HCS08.

Table 3.76 Tool Settings — RS08 Compiler > Language > CompactC++ Features Options

Option	Description
Disable all compactC++ features (-Cn)	If the -C++ option is enabled, you can disable the compactC++ features. • Vf : Virtual functions are not allowed. Avoid having virtual tables that consume a lot of memory. • Tpl : Templates are not allowed. Avoid having many generated functions perform similar operations. • Ptm : Pointer to member not allowed. Avoid having pointer-to-member objects that consume a lot of memory. • Mih : Multiple inheritance is not allowed. Avoid having complex class hierarchies. Because virtual base classes are logical only when used with multiple inheritance, they are also not allowed. • Ctr : The C++ Compiler can generate several kinds of functions, if necessary: – Default Constructor – Copy Constructor – Destructor – Assignment operator With this option enabled, the Compiler does not create those functions. This is useful when compiling C sources with the C++ Compiler, assuming you do not want C structures to acquire member functions. • Cpr : Class parameters and class returns are not allowed. Avoid overhead with Copy Constructor and Destructor calls when passing parameters, and passing return values of class type.
Do not allow virtual functions (-Cn=Vf)	Virtual functions are not allowed. Avoid having virtual tables that consume a lot of memory
Do not allow templates (-Cn=Tpl)	Templates are not allowed. Avoid having many generated functions perform similar operations.
Do not allow pointer to member (-Cn=Ptm)	Pointer to member not allowed. Avoid having pointer-to-member objects that consume a lot of memory.

Table 3.76 Tool Settings — RS08 Compiler > Language > CompactC++ Features Options

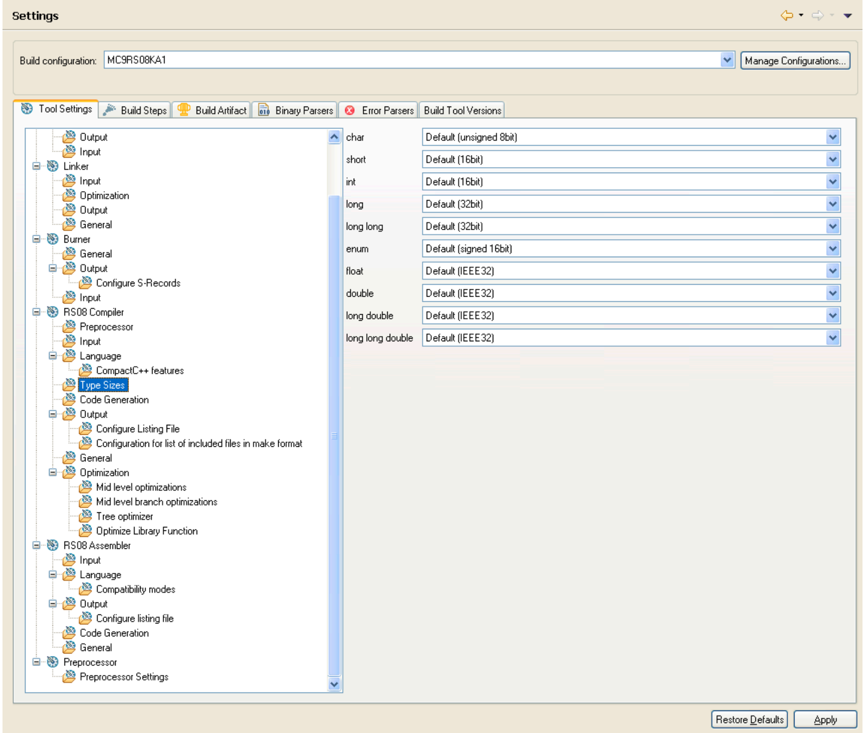
Option	Description
Do not allow multiple inheritance and virtual base classes (-Cn=Mih)	Multiple inheritance is not allowed. Avoid having complex class hierarchies. Because virtual base classes are logical only when used with multiple inheritance, they are also not allowed.
Do not create compiler defined functions (-Cn=Ctr)	The C++ Compiler can generate several kinds of functions, if necessary: • Default Constructor • Copy Constructor • Destructor • Assignment operator With this option enabled, the Compiler does not create those functions. This is useful when compiling C sources with the C++ Compiler, assuming you do not want C structures to acquire member functions.
Do not allow class parameters and class returns (-Cn=Ctr)	Class parameters and class returns are not allowed. Avoid overhead with Copy Constructor and Destructor calls when passing parameters, and passing return values of class type.

RS08 Compiler > Type Sizes

Use this panel to specify the available data type size options for the RS08 Compiler.

Figure 3.84 shows the **Type Sizes** panel.

Figure 3.84 Tool Settings — RS08 Compiler > Type Sizes


Settings
Build configuration:
MC9RS08KA1
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

char	Default (unsigned 8bit)
short	Default (16bit)
int	Default (16bit)
long	Default (32bit)
long long	Default (32bit)
enum	Default (signed 16bit)
float	Default (IEEE32)
double	Default (IEEE32)
long double	Default (IEEE32)
long long double	Default (IEEE32)

Table 3.77 lists and describes the type size options for RS08 Compiler.

Table 3.77 Tool Settings — RS08 Compiler > Type Sizes

Option	Description
char	Selects the size of the `char` type. Options are: • Default (unsigned 8bit) • unsigned 8bit (`-TuCC1`) • signed 8bit (`-TsCC1`) • signed 16bit (`-TsCC2`) • signed 32bit (`-TsCC4`)
short	Selects the size of the `short` type. Options are: • Default (16bit) • signed 8bit (`-TS1`) • signed 16bit (`-TS2`) • signed 32bit (`-TS4`)
int	Selects the size of the `int` type. Options are: • Default (16bit) • signed 8bit (`-TI1`) • signed 16bit (`-TI2`) • signed 32bit (`-TI4`)
long	Selects the size of the `long` type. Options are: • Default (32bit) • signed 8bit (`-TL1`) • signed 16bit (`-TL2`) • signed 32bit (`-TL4`)
long long	Selects the size of the `long long` type. Options are: • Default (32bit) • signed 8bit (`-TLL1`) • signed 16bit (`-TLL2`) • signed 32bit (`-TLL4`)

Table 3.77 Tool Settings — RS08 Compiler > Type Sizes (*continued*)

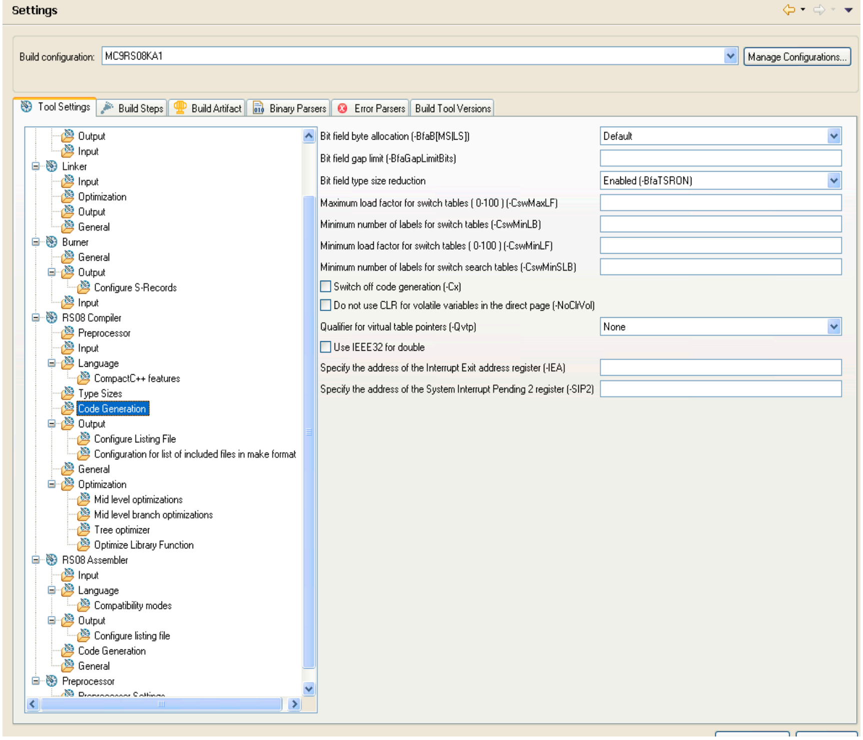
Option	Description
enum	Selects the size of the `enum` type. Options are: • Default (signed 16bit) • signed 8bit (`-TE1sE`) • signed 16bit (`-TE2sE`) • signed 32bit (`-TE4sE`) • unsigned 8bit (`-TE1uE`)
float	Selects the size of the `float` type. Options are: • Default (IEEE32) • IEEE32
double	Selects the size of the `double` type. Options are: • Default (IEEE32) • IEEE32
long double	Selects the size of the `long double` type. Options are: • Default (IEEE32) • IEEE32
long long double	Selects the size of the `long long double` type. Options are: • Default (IEEE32) • IEEE32

RS08 Compiler > Code Generation

Use this panel to specify code- and symbol-generation options for the RS08 Compiler

Figure 3.85 shows the **Code Generation** panel.

Figure 3.85 Tool Settings — RS08 Compiler > Code Generation


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor

Build configuration: MC9RS08KA1

Bit field byte allocation (-BfaB[MS|LS]) Default
Bit field gap limit (-BfaGapLimitBits)
Bit field type size reduction Enabled (-BfaTSRON)
Maximum load factor for switch tables (0-100) (-CswMaxLF)
Minimum number of labels for switch tables (-CswMinLB)
Minimum load factor for switch tables (0-100) (-CswMinLF)
Minimum number of labels for switch search tables (-CswMinSLB)
☐ Switch off code generation (-Cx)
☐ Do not use CLR for volatile variables in the direct page (-NoClrVol)
Qualifier for virtual table pointers (-Qvtp) None
☐ Use IEEE32 for double
Specify the address of the Interrupt Exit address register (-IEA)
Specify the address of the System Interrupt Pending 2 register (-SIP2)

Table 3.78 lists and describes the code generation options for RS08 compiler.

Table 3.78 Tool Settings — RS08 Compiler > Code Generation Options

Option	Description
Bit field byte allocation (-BfaB[MS|LS])	Normally, bits in byte bitfields are allocated from the least significant bit to the most significant bit. This produces less code overhead if a byte bitfield is allocated only partially.
Bit field gap limit (-BfaGapLimitBits)	Check to affect the maximum allowable number of gap bits. The bitfield allocation tries to avoid crossing a byte boundary whenever possible. To optimize accesses, the compiler may insert some padding or gap bits.

Table 3.78 Tool Settings — RS08 Compiler > Code Generation Options (*continued*)

Option	Description
Bit field type size reduction	This option is configurable whether or not the compiler uses type-size reduction for bitfields. Type-size reduction means that the compiler can reduce the type of an int bitfield to a char bitfield if it fits into a character. This allows the compiler to allocate memory only for one byte instead of for an integer. Options are: • **Enabled (-BfsTSRON)** • **Disabled (-BfsTSOFF)**
Maximum load factor for switch tables (0-100) (-CswMaxLF)	Allows changing the default strategy of the Compiler to use tables for switch statements; is only available if the compiler supports switch tables.
Minimum number of labels for switch tables (-CswMinLB)	Allows changing the default strategy of the Compiler using tables for switch statements; is only available if the compiler supports switch tables.
Minimum load factor for switch tables (0-100) (-CswMinLF)	Allows the Compiler to use tables for switch statements; is only available if the compiler supports switch tables.
Minimum number of labels for switch search tables (-CswMinSLB)	Allows the Compiler to use tables for switch statements. Using a search table improves code density, but the execution time increases. Every time an entry in a search table must be found, all previous entries must be checked first. For a dense table, the right offset is computed and accessed. In addition, note that all backends implement search tables (if at all) by using a complex runtime routine. This may make debugging more complex.
Switch off code generation (-Cx)	Disables the code generation process of the Compiler. No object code is generated, though the Compiler performs a syntactical check of the source code. This allows a quick test if the Compiler accepts the source without errors.

Table 3.78 Tool Settings — RS08 Compiler > Code Generation Options (*continued*)

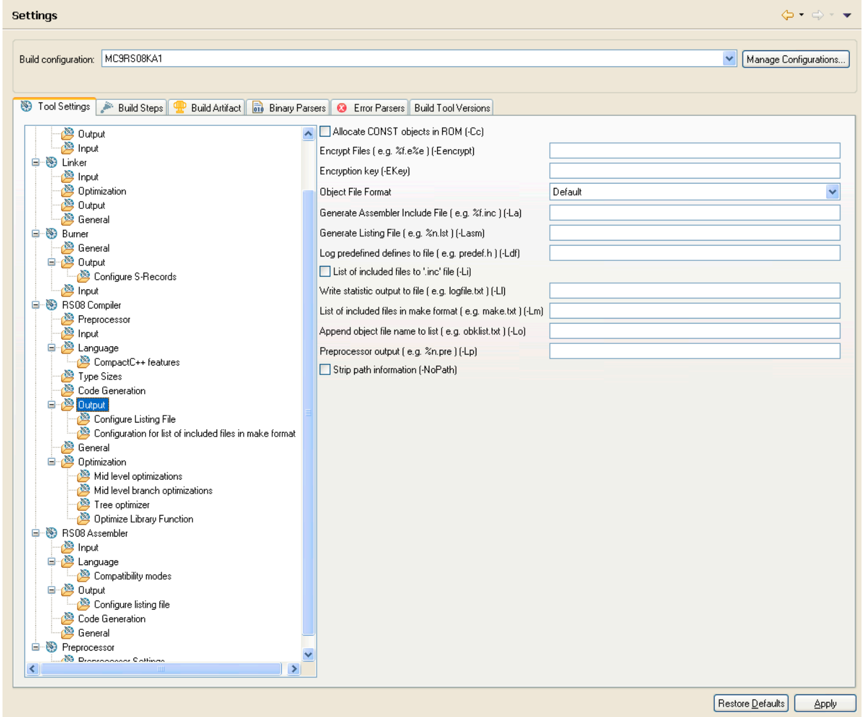
Option	Description
Do not use CLR for volatile variables in the direct page (-NoClrVol)	Inhibits the use of CLR for volatile variables in the direct page. The CLR instruction on HC08 has a read cycle. This may lead to unwanted lateral effects (e.g. if the variable is mapped over a hardware register).
Qualifier for virtual table pointers (-Qvtp)	Using a virtual function in C++ requires an additional pointer to virtual function tables. The Compiler cannot access the pointer and generates the pointer in every class object when virtual function tables are associated.
Use IEEE32 for double	Check to use IEEE32 for doubles instead of IEEE64 (default).
Specify the address of the Interrupt Exit address register (-IEA)	Specifies the address of the interrupt exit address register. By default, it is 0x200.
Specify the address of the System Interrupt Pending 2 register (-SIP2)	Specifies the address of the System Interrupt Pending 2 register. By default, it is set to 0x1D.

RS08 Compiler > Output

Use this panel to control how the compiler generates the output file, as well as error and warning messages. You can specify whether to allocate constant objects in ROM, generate debugging information, and strip file path information.

Figure 3.86 shows the **Output** panel.

Figure 3.86 Tool Settings — RS08 Compiler > Output



Build configuration: MC9RS08KA1

☐ Allocate CONST objects in ROM (-Cc)

Encrypt Files (e.g. %f.e%e) (-Eencrypt)

Encryption key (-EKey)

Object File Format: Default

Generate Assembler Include File (e.g. %f.inc) (-La)

Generate Listing File (e.g. %n.lst) (-Lasm)

Log predefined defines to file (e.g. predef.h) (-Ldf)

☐ List of included files to '.inc' file (-Li)

Write statistic output to file (e.g. logfile.txt) (-Ll)

List of included files in make format (e.g. make.txt) (-Lm)

Append object file name to list (e.g. obklist.txt) (-Lo)

Preprocessor output (e.g. %n.pre) (-Lp)

☐ Strip path information (-NoPath)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Table 3.79 lists and describes the output options for RS08 compiler.

Table 3.79 Tool Settings — RS08 Compiler > Output Options

Option	Description
Allocate CONST objects in ROM (-Cc)	Check to enables the Compiler assign const objects into the ROM_VAR segment, which the parameter file assigns to a ROM section.
Encrypt File (e.g. %f.e%e) (-Eencrypt)	Encrypts using the given key with the -Ekey: Encryption Key option.
Encryption key (-EKey)	Encrypt files with the given key number (-Eencrypt option).The default encryption key is 0. Using this default is not recommended.

Table 3.79 Tool Settings — RS08 Compiler > Output Options (*continued*)

Option	Description
Generate Assembler Include File (e.g. %f.inc) (-La)	Enables the Compiler to generate an assembler include file when the CREATE_ASM_LISTING pragma occurs. The name of the created file is specified by this option. If no name is specified, a default of %f.inc is taken. To put the file into the directory specified by the TEXTPATH: Text File Path environment variable, use the option -la=%n.inc. The %f option already contains the path of the source file. When %f is used, the generated file is in the same directory as the source file. The content of all modifiers refers to the main input file and not to the actual header file. The main input file is the one specified on the command line.
Generate Listing File (e.g. %n.lst) (-Lasm)	Enables the Compiler to generate an assembler listing file directly. The Compiler also prints all assembler-generated instructions to this file. The option specifies the name of the file. If no name is specified, the Compiler takes a default of %n.lst. If the resulting filename contains no path information the Compiler uses the TEXTPATH: Text File Path environment variable. The syntax does not always conform with the inline assembler or the assembler syntax. Therefore, use this option only to review the generated code. It cannot currently be used to generate a file for assembly.

Table 3.79 Tool Settings — RS08 Compiler > Output Options (*continued*)

Option	Description
Log predefined defines to file (e.g. predef.h) (-Ldf)	Enables the Compiler to generate a text file that contains a list of the compiler-defined #define. The default filename is predef.h, but may be changed (e.g., -Ldf="myfile.h"). The file is generated in the directory specified by the TEXTPATH: Text File Path environment variable. The defines written to this file depend on the actual Compiler option settings (e.g., type size settings or ANSI compliance). **Note:** The defines specified by the command line (-D: Macro Definition option) are not included. This option may be very useful for SQA. With this option it is possible to document every #define which was used to compile all sources. Note: This option only has an effect if a file is compiled. This option is unusable if you are not compiling a file.
List of included files to '.inc' file (-Li)	Enables the Compiler to generate a text file which contains a list of the #include files specified in the source. This text file shares the same name as the source file but with the extension, *.inc. The files are stored in the path specified by the TEXTPATH: Text File Path environment variable. The generated file may be used in make files.

Table 3.79 Tool Settings — RS08 Compiler > Output Options (*continued*)

Option	Description
Write statistic output to file (e.g. logfile.txt) (-Ll)	Enables the Compiler append statistical information about the compilation session to the specified file. The information includes Compiler options, code size (in bytes), stack usage (in bytes) and compilation time (in seconds) for each procedure of the compiled file. The Compiler appends the information to the specified filename (or the file make.txt, if no argument given). Set the TEXTPATH: Text File Path environment variable to store the file into the path specified by the environment variable. Otherwise the Compiler stores the file in the current directory.
List of included files in make format (e.g. logfile.txt)(-Lm)	This option causes the Compiler to generate a text file which contains a list of the #include files specified in the source. The generated list is in a make format. The -Lm option is useful when creating make files. The output from several source files may be copied and grouped into one make file. The generated list is in the make format. The filename does not include the path. After each entry, an empty line is added. The information is appended to the specified filename (or the make.txt file, if no argument is given).
Append object file name to list (e.g. obklist.txt)(-Lo)	This option causes the Compiler to append the object filename to the list in the specified file.The information is appended to the specified filename (or the file make.txt file, if no argument given).

Table 3.79 Tool Settings — RS08 Compiler > Output Options (*continued*)

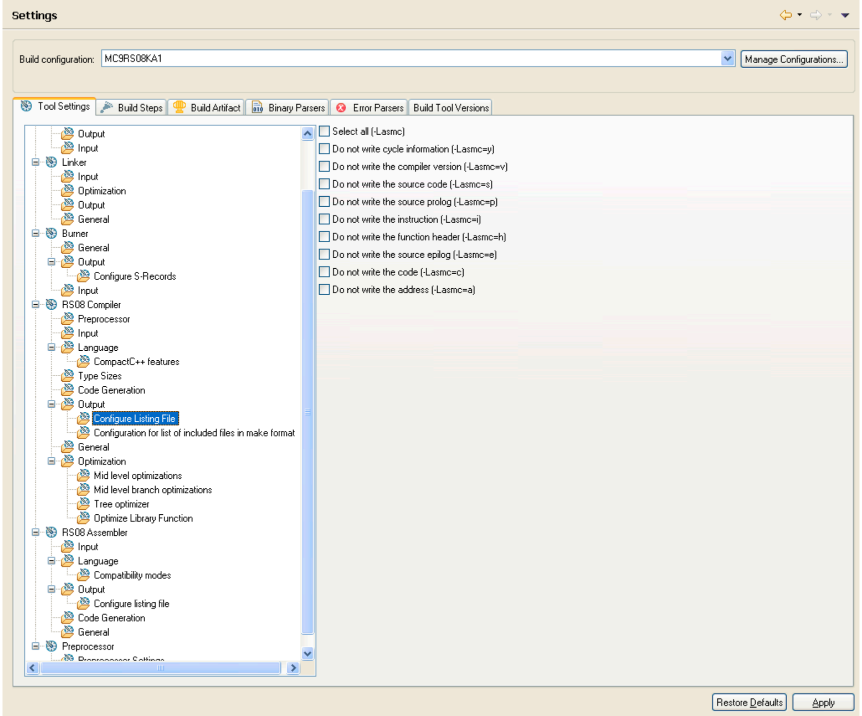
Option	Description
Preprocessor output (e.g. %n.pre)(-Lp)	This option causes the Compiler to generate a text file which contains the preprocessor's output. If no filename is specified, the text file shares the same name as the source file but with the extension, *.PRE (%n.pre). The TEXTPATH environment variable is used to store the preprocessor file.
Strip path information	Check to enable the compiler remove both unreferenced path reference from your program. This reduces your program's memory footprint.

RS08 Compiler > Output > Configure Listing File

Use this panel to configure the listing files for the RS08 Compiler to generate output.

Figure 3.87 shows the **Configure Listing File** panel.

Figure 3.87 Tools Settings — RS08 Compiler > Output > Configure Listing File


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Select all (-Lasmc)
☐ Do not write cycle information (-Lasmc=y)
☐ Do not write the compiler version (-Lasmc=v)
☐ Do not write the source code (-Lasmc=s)
☐ Do not write the source prolog (-Lasmc=p)
☐ Do not write the instruction (-Lasmc=i)
☐ Do not write the function header (-Lasmc=h)
☐ Do not write the source epilog (-Lasmc=e)
☐ Do not write the code (-Lasmc=c)
☐ Do not write the address (-Lasmc=a)

Table 3.80 lists and describes the Configure Listing FIle options for RS08 compiler.

Table 3.80 Tool Settings — RS08 Compiler > Output > Configure Listing File Options

Option	Description
Select all (-Lasmc)	This option configures the output format of the listing file generated with the *Generate Listing File* option. The addresses, the hex bytes, and the instructions are selectively switched off.
Do not write cycle information (-Lasmc=y)	This option switches off the cycle information from the output format of the listing file.
Do not write compiler version (-Lasmc=v)	This option switches off the compiler version from the output format of the listing file.

Table 3.80 Tool Settings — RS08 Compiler > Output > Configure Listing File Options

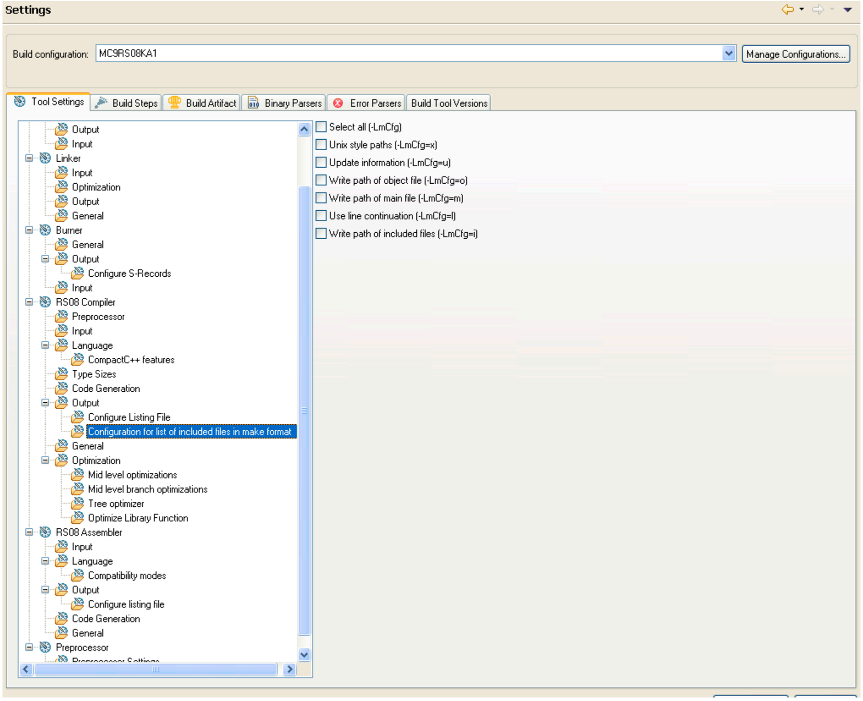
Option	Description
Do not write the source code (-Lasmc=s)	This option switches off the source code from the output format of the listing file.
Do not write source prolog (-Lasmc=p)	This option switches off the source prolog from the output format of the listing file.
Do not write the instruction (-Lasmc=i)	This option switches off the instruction from the output format of the listing file.
Do not write the function header (-Lasmc=h)	This option switches off the function header from the output format of the listing file.
Do not write source epilog (-Lasmc=e)	This option switches off the source epilog from the output format of the listing file.
Do not write the code (-Lasmc=c)	This option switches off the code from the output format of the listing file.
Do not write the address (-Lasmc=a)	This option switches off the address from the output format of the listing file.

RS08 Compiler > Output > Configuration for list of included files in make format

Use this panel to configure the list of included files in make format for the RS08 Compiler to generate the output.

Figure 3.88 shows the **Configuration for list of included files in make format** panel.

Figure 3.88 Tools Settings — RS08 Compiler > Output > Configuration for list of included files in make format


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings

Build configuration: MC9RS08KA1

☐ Select all (-LmCfg)
☐ Unix style paths (-LmCfg=x)
☐ Update information (-LmCfg=u)
☐ Write path of object file (-LmCfg=o)
☐ Write path of main file (-LmCfg=m)
☐ Use line continuation (-LmCfg=l)
☐ Write path of included files (-LmCfg=i)

Table 3.81 lists and describes the Configurfor list of included files in make format options for RS08 compiler.

Table 3.81 Tool Settings — RS08 Compiler > Output > Configure Listing File Options

Option	Description
Select all (-LmCfg)	This option is used when configuring the *List of Included Files in Make Format (-Lm)* option. The -LmCfg option is operative only if the -Lm option is also used. The -Lm option produces the ‘dependency’ information for a make file.
Unix style paths (-LmCfg=x)	Use this option to writes the path names in Unix style.

Table 3.81 Tool Settings — RS08 Compiler > Output > Configure Listing File Options

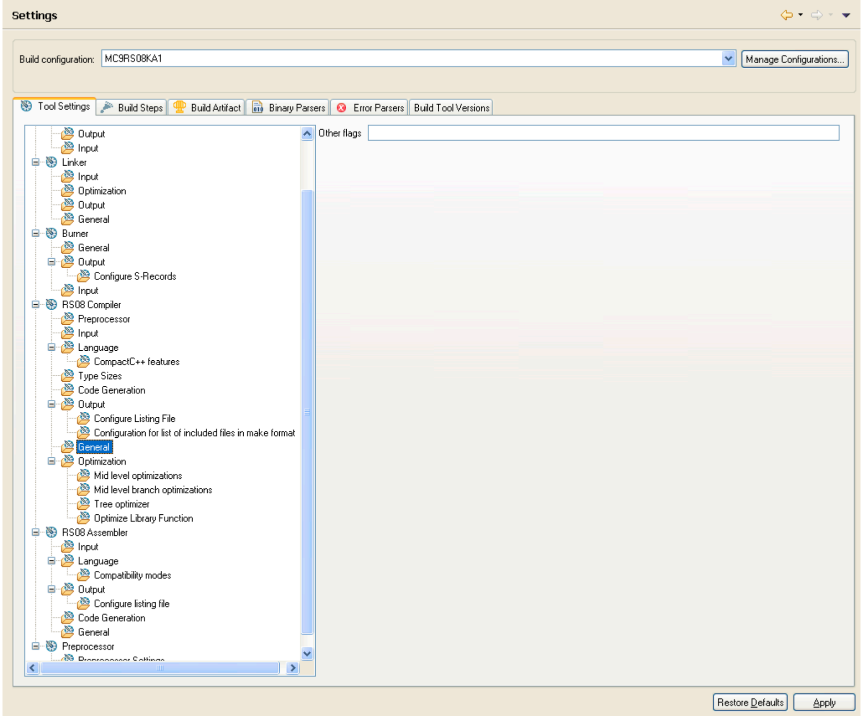
Option	Description
Update information (-LmCfg=u)	This option updates the information in the output file. If the file does not exist, the Compiler creates the file. If the file exists and the current information is not yet in the file, the Compiler appends the information to the file. If the information is already present, the Compiler updates the information. This allows you to specify this suboption for each compilation ensuring that the make dependency file is always up to date.
Write path of object file (-LmCfg=o)	This option writes the full name of the target object file.
Write path of main file (-LmCfg=m)	This option writes the full path of the compiled file. This is necessary when there are files with the same name in different directories.
Write line continuation (-LmCfg=l)	This option uses line continuation for each single entry in the dependency list. This improves readability.
Write path of included file (-LmCfg=i)	This option writes the full path of all included files in the dependency list.

RS08 Compiler > General

Use this panel to specify other flags for the RS08 Compiler to use.

Figure 3.89 shows the **General** panel.

Figure 3.89 Tool Settings — RS08 Compiler > General



Build configuration: MC9RS08KA1

Other flags:

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Output
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Restore Defaults
Apply

Table 3.82 lists and describes the general options for RS08 compiler.

Table 3.82 Tool Settings — RS08 Compiler > General Options

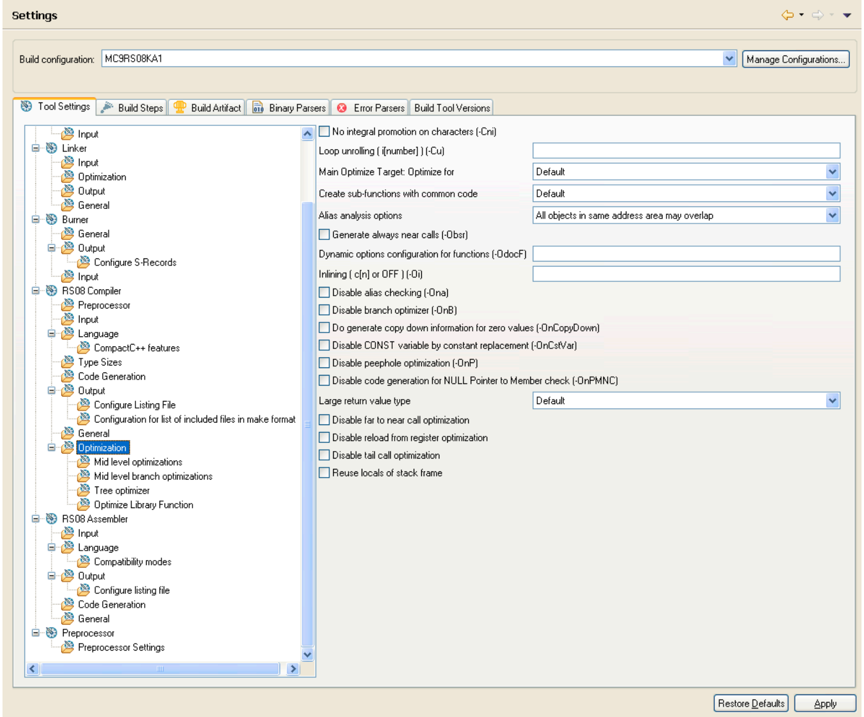
Option	Description
Other flags	Specify additional command line options for the compiler; type in custom flags that are not otherwise available in the UI.

RS08 Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.90 shows the **Optimization** panel.

Figure 3.90 Tool Settings — RS08 Compiler > Optimization


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ No integral promotion on characters (-Cni)

Loop unrolling (i[number]) (-Cu)

Main Optimize Target: Optimize for: Default

Create sub-functions with common code: Default

Alias analysis options: All objects in same address area may overlap

☐ Generate always near calls (-Obsr)

Dynamic options configuration for functions (-OdocF)

Inlining (c[n] or OFF) (-Oi)

☐ Disable alias checking (-Ona)

☐ Disable branch optimizer (-OnB)

☐ Do generate copy down information for zero values (-OnCopyDown)

☐ Disable CONST variable by constant replacement (-OnCstVar)

☐ Disable peephole optimization (-OnP)

☐ Disable code generation for NULL Pointer to Member check (-OnPMNC)

Large return value type: Default

☐ Disable far to near call optimization

☐ Disable reload from register optimization

☐ Disable tail call optimization

☐ Reuse locals of stack frame

Table 3.83 lists and describes the optimization options for RS08 compiler.

Table 3.83 Tool Settings — RS08 Compiler > Optimization Options

Option	Description
No integral promotion on characters (-Cni)	Enhances character operation code density by omitting integral promotion. This option enables behavior that is not ANSI-C compliant. Code generated with this option set does not conform to ANSI standards. Code compiled with this option is not portable. Using this option is not recommended in most cases.
Loop unrolling (i[number]) (-Cu)	Enables loop unrolling with the following restrictions: • Only simple for statements are unrolled, e.g., for (i=0; i<10; i++) • Initialization and test of the loop counter must be done with a constant. • Only <, >, <=, >= are permitted in a condition. • Only ++ or -- are allowed for the loop variable increment or decrement. • The loop counter must be integral. • No change of the loop counter is allowed within the loop. • The loop counter must not be used on the left side of an assignment. • No address operator (&) is allowed on the loop counter within the loop. • Only small loops are unrolled: Loops with few statements within the loop. Loops with fewer than 16 increments or decrements of the loop counter. The bound may be changed with the optional argument = i<number>. The -Cu=i20 option unrolls loops with a maximum of 20 iterations.

Table 3.83 Tool Settings — RS08 Compiler > Optimization Options (*continued*)

Option	Description
Main Optimize Target: Optimize for	There are various points where the Compiler has to select between two possibilities: it can either generate fast, but large code, or small but slower code. The Compiler generally optimizes on code size. It often has to decide between a runtime routine or an expanded code. The programmer can decide whether to select between the slower and shorter or the faster and longer code sequence by setting a command line switch. • The **Code Size (-Os)** option directs the Compiler to optimize the code for smaller code size. The Compiler trades faster-larger code for slower-smaller code. • The **Execution Time (-Ot)** option directs the Compiler to optimize the code for faster execution time. The Compiler replaces slower/smaller code with faster/larger code. This option only affects some special code sequences. This option has to be set together with other optimization options (e.g., register optimization) to get best results.
Create sub-functions with common code	Performs the reverse of inlining. It detects common code parts in the generated code. The Compiler moves the common code to a different place and replaces all occurrences with a JSR to the moved code. At the end of the common code, the Compiler inserts an RTS instruction. The Compiler increases all SP uses by an address size. This optimization takes care of stack allocation, control flow, and of functions having arguments on the stack. Inline assembler code is never treated as common code. Options are: • **Default** • **Disable (-Onf)** • **Enable (-Of)**

Table 3.83 Tool Settings — RS08 Compiler > Optimization Options (*continued*)

Option	Description
Alias analysis options	These four different options allow the programmer to control the alias behavior of the compiler. The option -oaaddr is the default because it is safe for all C programs. Use option -oaansi if the source code follows the ANSI C99 alias rules. If objects with different types never overlap in your program, use option -oatype. If your program doesn't have aliases at all, use option -oanone (or the ICG option -ona, which is supported for compatibility reasons).
Generate always near calls (-Obsr)	This option forces the compiler to always generate near calls, i.e. use BSR instruction instead of a JSR in order to reduce code size. Without this option the compiler checks the range of the call to determine if a BSR can be generated instead of a JSR.
Dynamic options configuration for functions (-OdocF)	Allows the Compiler to select from a set of options to reach the smallest code size for every function. Without this feature, you must set fixed Compiler switches over the whole compilation unit. With this feature, the Compiler finds the best option combination from a user-defined set for every function.
Inlining (C[n] or OFF) (-Oi)	Enables inline expansion. If there is a #pragma INLINE before a function definition, all calls of this function are replaced by the code of this function, if possible. Using the -Oi=c0 option switches off inlining. Functions marked with the #pragma INLINE are still inlined. To disable inlining, use the -Oi=OFF option.
Disable alias checking (-Ona)	Prevents the Compiler from redefining these variables, which lets you reuse already-loaded variables or equivalent constants. Use this option only when you are sure no real writes of aliases to a variable memory location will occur.
Disable branch optimizer (-OnB)	Disables all branch optimizations.

Table 3.83 Tool Settings — RS08 Compiler > Optimization Options (*continued*)

Option	Description
Do generate copy down information for zero values (-OnCopyDown)	Restricts the compiler from generating a copy down for i. The initialization with zero optimization shown for the arr array only works in the HIWARE format. The ELF format requires initializing the whole array to zero.
Disable CONST variable by constant replacement (-OnCsfVar)	Lets you switch OFF the replacement of CONST variable by the constant value.
Disable peephole optimization (-OnP)	Disables the whole peephole optimizer. To disable only a single peephole optimization, use the optional syntax -OnP=<char>.
Disable code generation for NULL Pointer to Member check (-OnPMNC)	Before assigning a pointer to a member in C++, you must ensure that the pointer to the member is not NULL in order to generate correct and safe code. In embedded systems development, the difficulty becomes generating the denser code while avoiding overhead whenever possible (this NULL check code is a good example). This option enables you to switch off the code generation for the NULL check.
Large return value type	Compiler supports this option even though returning a 'large' return value may be not as efficient as using an additional pointer. The Compiler introduces an additional parameter for the return value if the return value cannot be passed in registers. Options are: • **Default** • **Large return value pointer, always with temporary (-Rpt)** • **Large return value pointer and temporary elimination (-Rpe)**
Disable far to near optimization	Disables the JSR to BSR optimization. The compiler checks the range of the call to determine if a BSR can be generated instead of a JSR. If -Onbsr is used this optimization will be disabled.
Disable reload from register optimization	Disables the low level register trace optimizations. If you use the option the code becomes more readable, but less optimal.

Table 3.83 Tool Settings — RS08 Compiler > Optimization Options (*continued*)

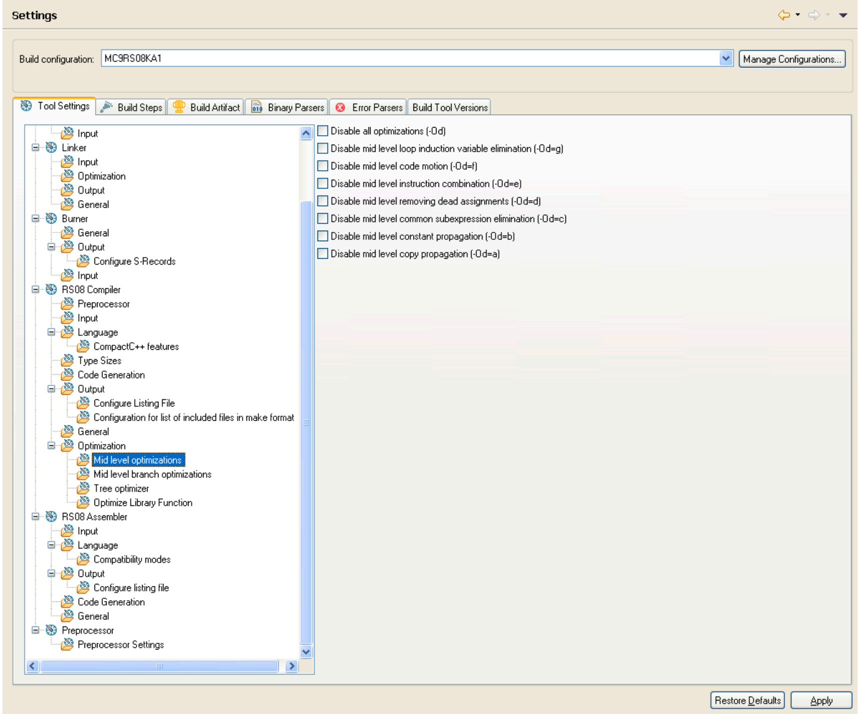
Option	Description
Disable tail call optimizations	Allows the compiler to remove all the entry and exit code from the current function.By default, the compiler replaces trailing calls (JSR/BSR) with JMP instructions if the function does not contain any other function calls.
Reuse locals of stack frame	Instructs the compiler to reuse the location of local variables/temporaries whenever possible. When used, the compiler analyzes which local variables are alive simultaneously. Based on that analysis the compiler selects the best memory layout for for variables. Two or more variables may end up sharing the same memory location.

RS08 Compiler > Optimization > Mid level optimizations

Use this panel to configure the Mid level optimization options for the RS08 compiler.

Figure 3.91 shows the **Mid level optimizations** settings.

Figure 3.91 Tool Settings — RS08 Compiler > Optimization > Mid level optimizations


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Disable all optimizations (-Od)
☐ Disable mid level loop induction variable elimination (-Od=g)
☐ Disable mid level code motion (-Od=f)
☐ Disable mid level instruction combination (-Od=e)
☐ Disable mid level removing dead assignments (-Od=d)
☐ Disable mid level common subexpression elimination (-Od=c)
☐ Disable mid level constant propagation (-Od=b)
☐ Disable mid level copy propagation (-Od=a)

Table 3.84 lists and describes the Mid level optimizations options for RS08 compiler.

Table 3.84 Tool Settings — RS08 Compiler > Optimization > Mid level optimizations

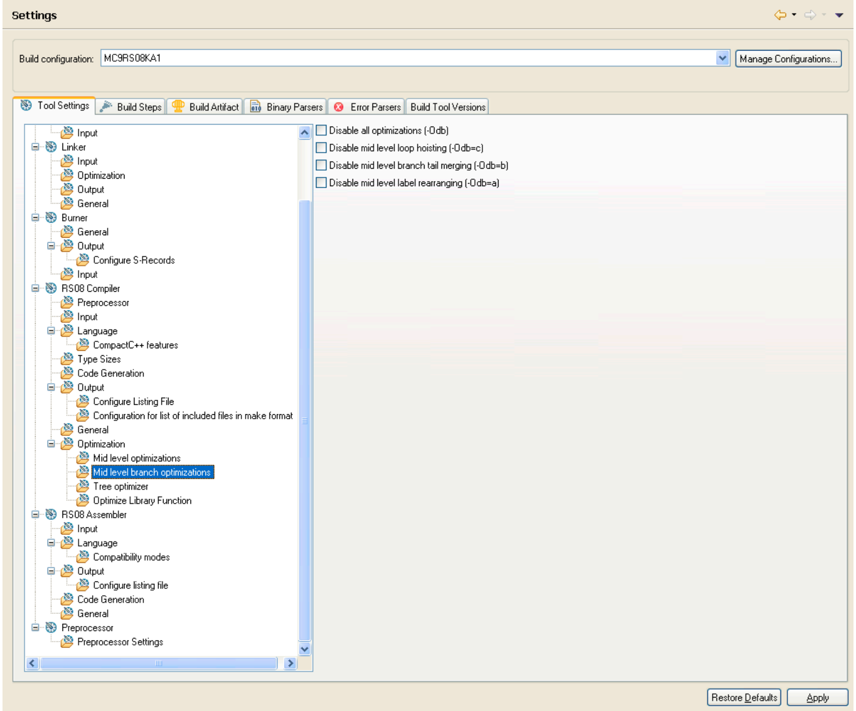
Option	Description
Disable all optimizations (-Od)	The backend of this compiler is based on the second generation intermediate code generator (SICG). All intermediate language and processor independent optimizations (cf. NULLSTONE) are performed by the SICG optimizer using the powerful static single assignment form (SSA form). The optimizations are switched off using -od. Currently four optimizations are implemented. This option disables all the optimizations.
Disable mid level loop induction variable elimination (-Od=g)	This option disables all the mid level loop induction variable elimination.
Disable mid level code motion (-Od=f)	This option disables all the mid level code motion.
Disable mid level instruction combination (-Od=e)	This option disables all the mid level instruction combination.
Disable mid level removing dead assignments (-Od=d)	This option disables removing dead assignments only.
Disable mid level common subexpression elimination (-Od=c)	This option disables removing dead assignments and CSE.
Disable mid level constant propagation (-Od=b)	This option disables mid level constant propagation.
Disable mid level copy propagation (-Od=a)	This option disables mid level copy propagation

RS08 Compiler > Optimization > Mid level branch optimizations

Use this option to specify the mid level branch optimization options.

Figure 3.92 shows the **Mid level branch optimizations** settings.

Figure 3.92 Tool Settings — RS08 Compiler > Optimization > Mid level branch optimizations


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Disable all optimizations (-Odb)
☐ Disable mid level loop hoisting (-Odb=c)
☐ Disable mid level branch tail merging (-Odb=b)
☐ Disable mid level label rearranging (-Odb=a)

[Table 3.85](#) lists and describes the Mid level branch optimizations options for RS08 compiler.

Table 3.85 Tool Settings — RS08 Compiler > Optimization > Mid level branch optimizations

Option	Description
Disable all optimizations (-Odb)	This option disables branch optimizations on the SSA form based on control flows. Label rearranging sorts all labels of the control flow to generate a minimum amount of branches.
Disable mid level loop hoisting (-Odb=c)	This option disables mid level loop hoisting.

Table 3.85 Tool Settings — RS08 Compiler > Optimization > Mid level branch optimizations (*continued*)

Option	Description
Disable mid level branch tail merging (-Odb=b)	This option disables only branch tail merging.
Disable mid level label rearranging (-Odb=a)	This option disables mid level label rearranging.

RS08 Compiler > Optimization > Tree optimizer

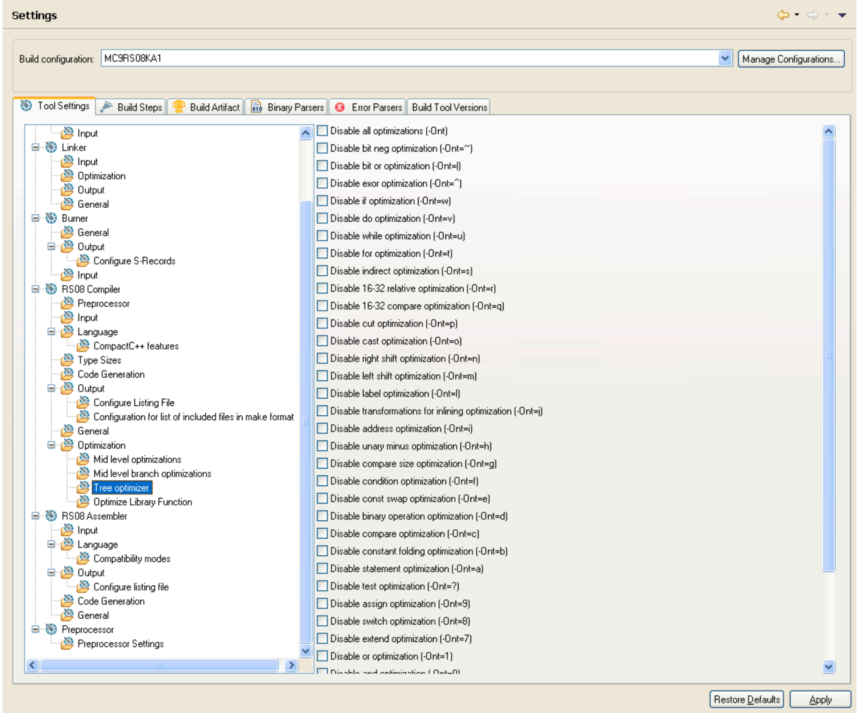
The Compiler contains a special optimizer which optimizes the internal tree data structure. This tree data structure holds the semantic of the program and represents the parsed statements and expressions.

This option disables the tree optimizer. This may be useful for debugging and for forcing the Compiler to produce ‘straightforward’ code.

Use this panel to configure the tree optimizer options for the RS08 compiler.

Figure 3.93 shows the **Tree optimizer** settings.

Figure 3.93 Tool Settings — RS08 Compiler > Optimization > Tree optimizer


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Disable all optimizations (-Ont)
☐ Disable bit neg optimization (-Ont=~)
☐ Disable bit or optimization (-Ont=|)
☐ Disable exor optimization (-Ont=^)
☐ Disable if optimization (-Ont=w)
☐ Disable do optimization (-Ont=v)
☐ Disable while optimization (-Ont=u)
☐ Disable for optimization (-Ont=t)
☐ Disable indirect optimization (-Ont=s)
☐ Disable 16-32 relative optimization (-Ont=r)
☐ Disable 16-32 compare optimization (-Ont=q)
☐ Disable cut optimization (-Ont=p)
☐ Disable cast optimization (-Ont=o)
☐ Disable right shift optimization (-Ont=n)
☐ Disable left shift optimization (-Ont=m)
☐ Disable label optimization (-Ont=l)
☐ Disable transformations for inlining optimization (-Ont=j)
☐ Disable address optimization (-Ont=i)
☐ Disable unary minus optimization (-Ont=h)
☐ Disable compare size optimization (-Ont=g)
☐ Disable condition optimization (-Ont=f)
☐ Disable const swap optimization (-Ont=e)
☐ Disable binary operation optimization (-Ont=d)
☐ Disable compare optimization (-Ont=c)
☐ Disable constant folding optimization (-Ont=b)
☐ Disable statement optimization (-Ont=a)
☐ Disable test optimization (-Ont=?)
☐ Disable assign optimization (-Ont=9)
☐ Disable switch optimization (-Ont=8)
☐ Disable extend optimization (-Ont=7)
☐ Disable or optimization (-Ont=1)

Table 3.86 lists and describes the Tree optimizer options for RS08 compiler.

Table 3.86 Tool Settings — RS08 Compiler > Optimization > Tree optimizer

Option	Description
Disable all optimizations (-Ont)	Disable all the optimizations.
Disable bit neg optimization (-Ont=~)	Disable optimization of ‘~~i’ into ‘i’.
Disable bit or optimization (-Ont=I)	Disable optimization of ‘il0xffff’ into ‘0xffff’.
Disable exor optimization (-Ont=^)	Disable optimization of ‘i^0’ into ‘i’.
Disable if optimization (-Ont=w)	Disable optimization of ‘if (1) i = 0;’ into ‘i = 0;’.
Disable do optimization (-Ont=v)	Disable optimization of ‘do ... while(0) into ‘...’.

Table 3.86 Tool Settings — RS08 Compiler > Optimization > Tree optimizer

Option	Description
Disable while optimization (-Ont=u)	Disable optimization of ‘while(1) ...;’ into ‘...;’.
Disable for optimization (-Ont=t)	Disable optimization of ‘for(;;) ...’ into ‘while(1) ...’.
Disable indirect optimization (-Ont=s)	Disable optimization of ‘*&i’ into ‘i’.
Disable 16-32 relative optimization (-Ont=r)	Disable optimization of ‘L<=4’ into 16-bit compares if 16-bit compares are better.
Disable 16-32 compare optimization (-Ont=q)	Reduction of long compares into int compares if int compares are better: (-Ont=q to disable it).
Disable cut optimization (-Ont=p)	Disable optimization of ‘(char)(long)i’ into ‘(char)i’.
Disable cast optimization (-Ont=o)	Disable optimization of ‘(short)(int)L’ into ‘(short)L’ if short and int have the same size.
Disable right shift optimization (-Ont=n)	Optimization of shift optimizations (<<, -Ont=n to disable it)
Disable left shift optimization (-Ont=m)	Optimization of shift optimizations (>>, -Ont=m to disable it)
Disable label optimization (-Ont=l)	Disable optimization removal of labels if not used.
Disable transformations for inlining optimization (-Ont=j)	This optimization transforms the syntax tree into an equivalent form in which more inlining cases can be done. This option only has an effect when inlining is enabled.
Disable address optimization (-Ont=i)	Disable optimization of ‘&*p’ into ‘p’.
Disable unary minus optimization (-Ont=h)	Disable optimization of ‘-(-i)’ into ‘i’.
Disable compare size optimization (-Ont=g)	
Disable condition optimization (-Ont=f)	Disable optimization of ‘(a==0)’ into ‘(!a)’.
Disable const swap optimization (-Ont=e)	Disable optimization of ‘2*i’ into ‘i*2’.

Table 3.86 Tool Settings — RS08 Compiler > Optimization > Tree optimizer

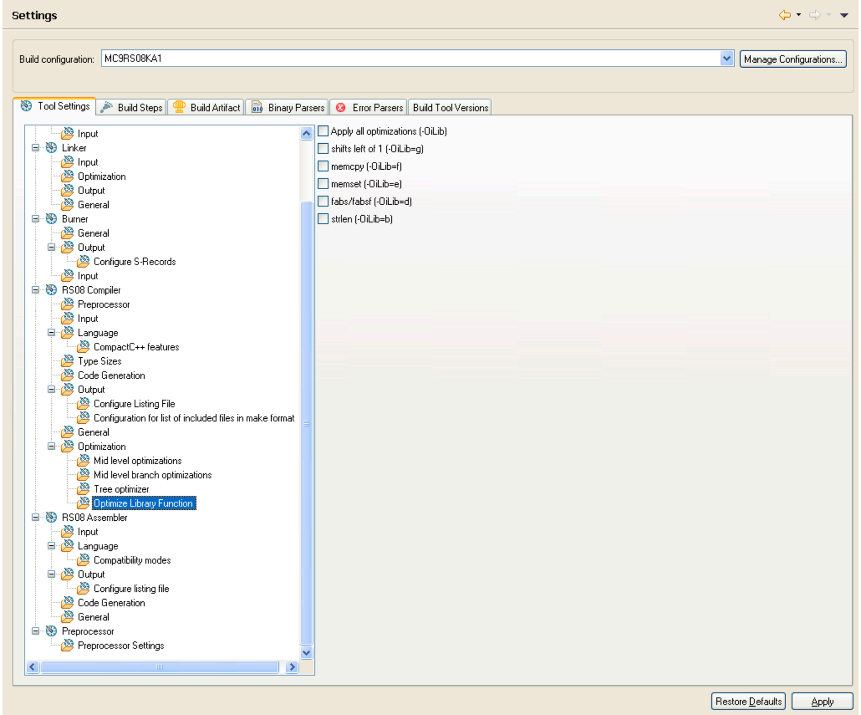
Option	Description
Disable binary operation optimization (-Ont=d)	Disable optimization of ‘us & ui’ into ‘us & (unsigned short) ui’.
Disable compare optimization (-Ont=c)	Disable optimization of ‘if ((long)i)’ into ‘if (i)’.
Disable constant folding optimization (-Ont=b)	Disable optimization of ‘3+7’ into ‘10’.
Disable statement optimization (-Ont=a)	Disable optimization of last statement in function if result is not used.
Disable test optimization (-Ont=?)	Disable optimization of ‘i = (int)(cond ? L1:L2);’ into ‘i = cond ? (int)L1:(int)L2;’.
Disable assign optimization (-Ont=9)	Disable optimization of ‘i=i;’.
Disable switch optimization (-Ont=8)	Disable optimization of empty switch statement.
Disable extend optimization (-Ont=7)	Disable optimization of ‘(long)(char)L’ into ‘L’.
Disable or optimization (-Ont=1)	Disable optimization of ‘a || 0’ into ‘a’.
Disable and optimization (-Ont=0)	Disable optimization of ‘a && 1’ into ‘a’.
Disable div optimization (-Ont=/)	Disable optimization of ‘a/1’ into ‘a’.
Disable minus optimization (-Ont=-)	Disable optimization of ‘a-0’ into ‘a’.
Disable plus optimization (-Ont=+)	Disable optimization of ‘a+0’ into ‘a’.
Disable mul optimization (-Ont=*)	Disable optimization of ‘a*1’ into ‘a’.
Disable bit and optimization (-Ont=)	Disable optimization of ‘a&0’ into ‘0’.
Disable mod optimization (-Ont=%)	Disable optimization of ‘a%1’ into ‘0’.

RS08 Compiler > Optimization > Optimize Library Function

This option enables the compiler to optimize specific known library functions to reduce execution time. The Compiler frequently uses small functions such as strcpy(), strcmp(), and so forth. Use this panel to configure the optimize library function options for the RS08 compiler.

Figure 3.94 shows the **Optimize Library Function** settings.

Figure 3.94 Tool Settings — RS08 Compiler > Optimization > Optimize Library Function


Settings
Build configuration: MC9RS08KA1
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

☐ Apply all optimizations (-OiLib)
☐ shifts left of 1 (-OiLib=g)
☐ memcpy (-OiLib=f)
☐ memset (-OiLib=e)
☐ fabs/fabsf (-OiLib=d)
☐ strlen (-OiLib=b)

Table 3.87 lists and describes the Mid level branch optimizations options for RS08 compiler.

Table 3.87 Tool Settings — RS08 Compiler > Optimization > Mid level branch optimizations

Option	Description
Apply all optimizations (-OiLib)	This option applies all the optimizations.
shifts left of 1 (-OiLib=g)	This option replace shifts left of 1 by array lookup.
memcpy (-OiLib=f)	This option inline calls to the memcpy() function.
memset (-OiLib=e)	This option inline calls to the memset() function.

Table 3.87 Tool Settings — RS08 Compiler > Optimization > Mid level branch optimizations (*continued*)

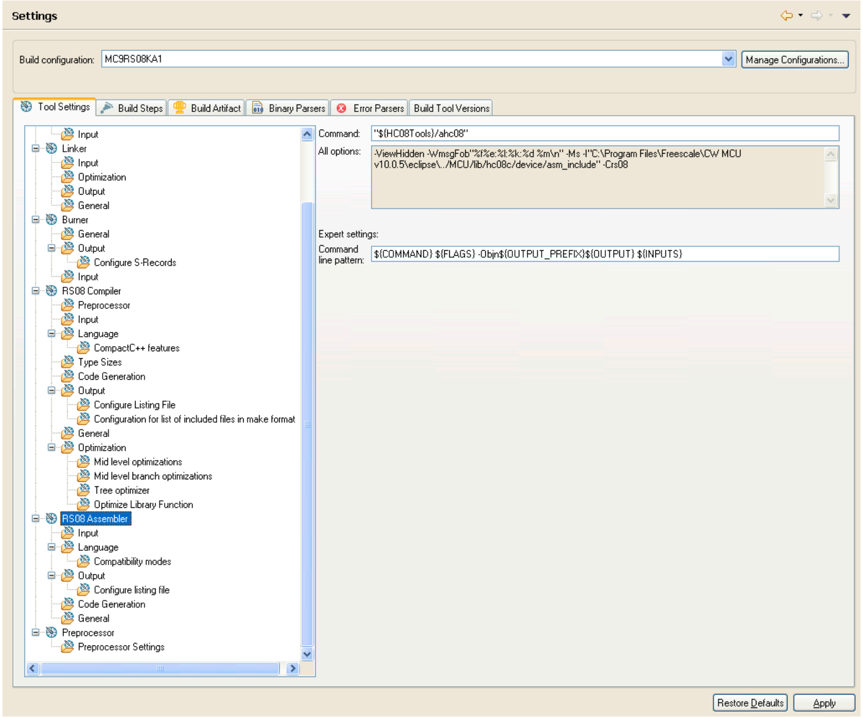
Option	Description
fabs/fabsf (-OiLib=d)	This option inline calls to the fabs() or fabsf() functions.
strlen (-OiLib=b)	This option inline calls to the strlen() function.

RS08 Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler.

Figure 3.95 shows the **Assembler** settings.

Figure 3.95 Tool Settings — RS08 Assembler


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Command: "${HC08Tools}/ahc08"

All options: -ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -Ms -I"C:\Program Files\Freescale\CW MCU v10.0.5\eclipse\../MCU/lib/hc08c/device/asm_include" -Crs08

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} -Objn${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.88 lists and describes the assembler options for RS08.

Table 3.88 Tool Settings — Assembler Options

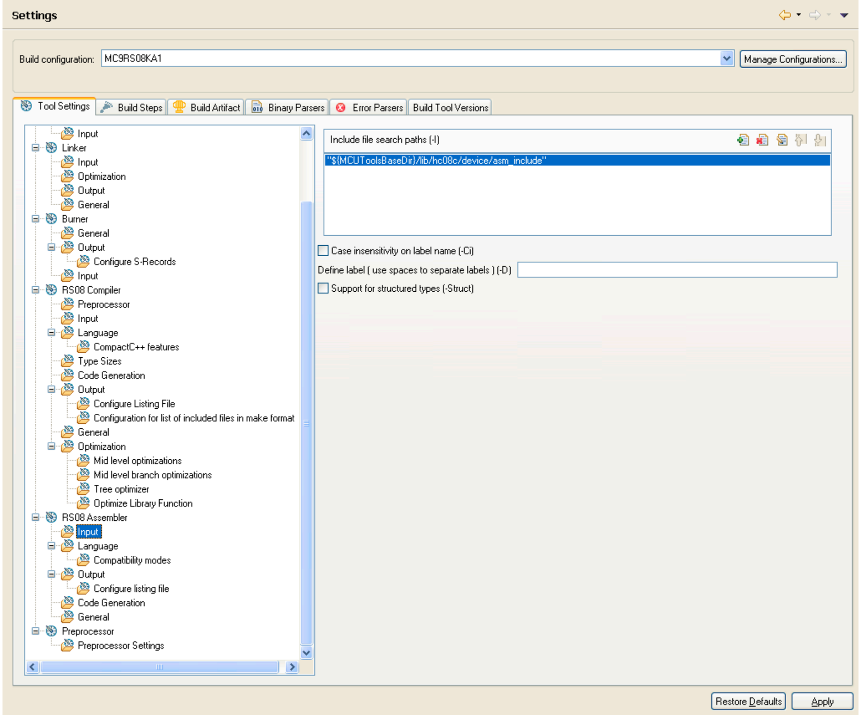
Option	Description
Command	Shows the location of the assembler executable file. Default value is: `"${HC08Tools}/ahc08.exe"`
All options	Shows the actual command line the assembler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${COMMAND} ${FLAGS}-Objn${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

RS08 Assembler > Input

Use this panel to specify file search paths and any additional include files the **RS08 Assembler** should use. You can specify multiple search paths and the order in which you want to perform the search.

Figure 3.96 shows the **Input** panel.

Figure 3.96 Tool Settings — RS08 Assembler > Input



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Include file search paths (-I)
"${MCUToolsBaseDir}/lib/hc08c/device/asm_include"

☐ Case insensitivity on label name (-Ci)

Define label (use spaces to separate labels) (-D)

☐ Support for structured types (-Struct)

Table 3.89 lists and describes the input options of RS08 assembler.

Table 3.89 Tool Settings — Assembler > Input options

Button	Description
Include file search paths (-I)	Lists the included file search paths.
Case sensitivity on label names (-Ci)	Check to make the label names case sensitive.
Define label (use spaces to separate labels) (-D)	Define labels that have to be included in the RS08 assembler input.
Support for structured types (-Struct)	Check to include the support for structured types.

Table 3.90 lists and describes the toolbar buttons that help work with the file search paths.

Table 3.90 Search Paths Toolbar Buttons

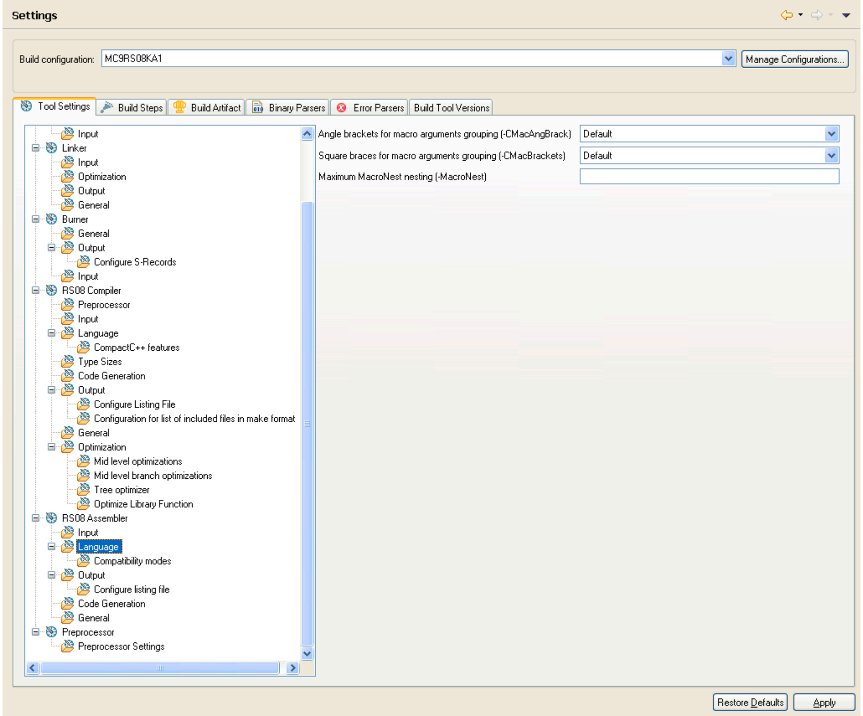
Button	Description
➕	**Add —** Click to open the **Add directory path** dialog box (Figure 3.64) and specify the file search path.
✖	**Delete —** Click to delete the selected file search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.65) and update the selected object file search path.
⇧	**Move up —** Click to move the selected file search path one position higher in the list.
⇩	**Move down —** Click to move the selected file search path one position lower in the list.

RS08 Assembler > Language

Use this panel to specify code- and symbol-generation options for the RS08 Compiler.

Figure 3.97 shows the **Language** panel.

Figure 3.97 Tool Settings — RS08 Assembler > Language


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Angle brackets for macro arguments grouping (-CMacAngBrack)	Default
Square braces for macro arguments grouping (-CMacBrackets)	Default
Maximum MacroNest nesting (-MacroNest)	

Table 3.91 lists and describes the language options for RS08 Assembler.

Table 3.91 Tool Settings — RS08 Assembler > Language Options

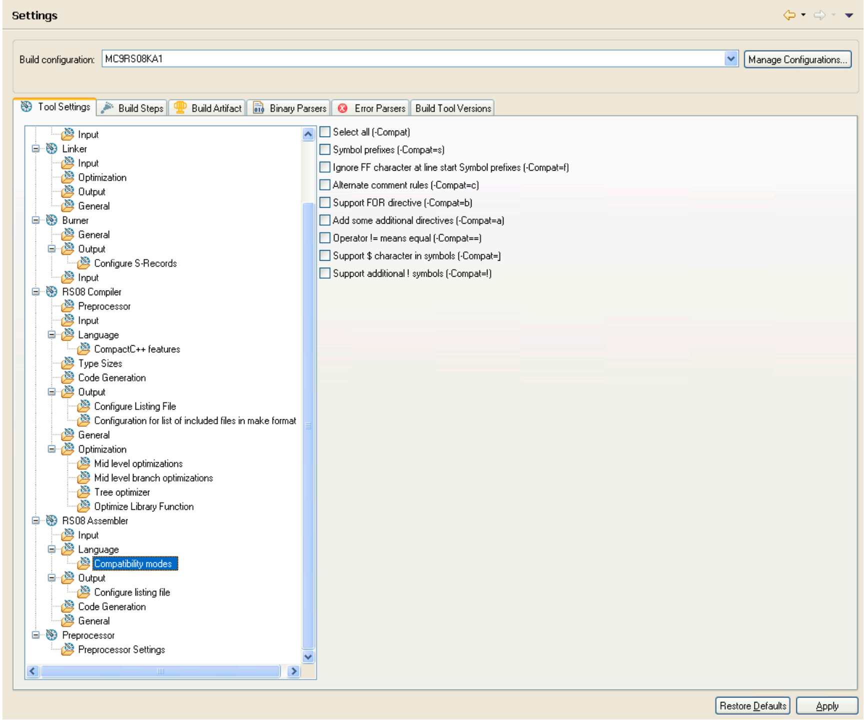
Option	Description
Angle brackets for macro arguments grouping (-CMacAngBrack)	Controls whether the < > syntax for macro invocation argument grouping is available. When it is disabled, the Assembler does not recognize the special meaning for < in the macro invocation context. There are cases where the angle brackets are ambiguous. In new code, use the [? ?] syntax instead. Options are: • **Allow** • **Disallow**
Square braces for macro arguments grouping (-CMacBrackets)	Controls the availability of the [? ?] syntax for macro invocation argument grouping. When it is disabled, the Assembler does not recognize the special meaning for [?] in the macro invocation context. Options are: • **Allow** • **Disallow**
Maximum MacroNest nesting (-MacroNest)	Controls how deep macros calls can be nested. Its main purpose is to avoid endless recursive macro invocations.

RS08 Assembler > Language > Compatibility modes

Use this panel to specify the compatibility modes options of the RS08 assembler.

Figure 3.98 shows the **Compatibility modes** panel.

Figure 3.98 Tool Settings — RS08 Assembler > Compatibility modes


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Select all (-Compat)
☐ Symbol prefixes (-Compat=s)
☐ Ignore FF character at line start Symbol prefixes (-Compat=f)
☐ Alternate comment rules (-Compat=c)
☐ Support FOR directive (-Compat=b)
☐ Add some additional directives (-Compat=a)
☐ Operator != means equal (-Compat==)
☐ Support $ character in symbols (-Compat=)
☐ Support additional ! symbols (-Compat=!)

Table 3.93 lists and describes the compatibility mode options for RS08 Assembler.

Table 3.92 Tool Settings — RS08 Assembler > Compatibility modes Options

Option	Description
Select all (-Compat)	
Symbol prefixes (-Compat=s)	With this suboption, the Assembler accepts "pgz:" and "byte:" prefixed for symbols in XDEFs and XREFs. They correspond to XREF.B or XDEF.B with the same symbols without the prefix.
Ignore FF character at line start Symbol prefixes (-Compat=f)	With this suboption, an otherwise improper character recognized from feed character is ignored.

Table 3.92 Tool Settings — RS08 Assembler > Compatibility modes Options (*continued*)

Option	Description
Alternate comment rules (-Compat=c)	With this suboption, comments implicitly start when a space is present after the argument list. A special character is not necessary. Be careful with spaces when this option is given because part of the intended arguments may be taken as a comment. However, to avoid accidental comments, the Assembler does issue a warning if such a comment does not start with a "*" or a ";".
Support FOR directive (-Compat=b)	With this suboption, the Assembler supports a FOR - Repeat assembly block assembly directive to generate repeated patterns more easily without having to use recursive macros.
Add some additional directives (-Compat=a)	With this suboption, some additional directives are added for enhanced compatibility. The Assembler actually supports a SECT directive as an alias of the usual SECTION - Declare Relocatable Section assembly directive. The SECT directive takes the section name as its first argument.
Operator != means equal (-Compat==)	The Assembler takes the default value of the != operator as not equal, as it is in the C language. For compatibility, this behavior can be changed to equal with this option. Because of the risks involved with this option for existing code, a message is issued for every != which is treated as equal.

Table 3.92 Tool Settings — RS08 Assembler > Compatibility modes Options (*continued*)

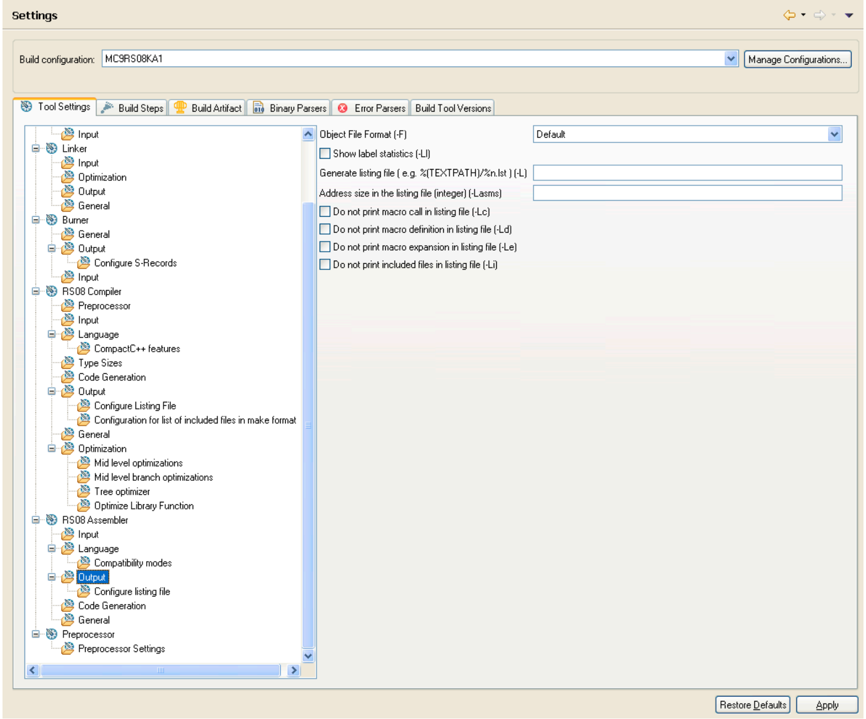
Option	Description
Support $ character in symbols (-Compat=)	With this suboption, the Assembler supports to start identifiers with a $ sign.
Support additional ! symbols (-Compat=!)	The following additional operators are defined when this option is used: • !^: exponentiation • !m: modulo • !@: signed greater or equal • !g: signed greater • !%: signed less or equal • !t: signed less than • !$: unsigned greater or equal • !S: unsigned greater • !&: unsigned less or equal • !l: unsigned less • !n: one complement • !w: low operator • !h: high operator **Note:** The default values for the following ! operators are defined: • !.: binary AND • !x: exclusive OR • !+: binary OR

RS08 Assembler > Output

Use this panel to control how the assembler generates the output file, as well as error and warning messages. You can specify whether to allocate constant objects in ROM, generate debugging information, and strip file path information.

Figure 3.99 shows the **Output** panel.

Figure 3.99 Tool Settings — RS08 Assembler > Output


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Object File Format (-F) Default

☐ Show label statistics (-Ll)

Generate listing file (e.g. %(TEXTPATH)/%n.lst) (-L)

Address size in the listing file (integer) (-Lasms)

☐ Do not print macro call in listing file (-Lc)

☐ Do not print macro definition in listing file (-Ld)

☐ Do not print macro expansion in listing file (-Le)

☐ Do not print included files in listing file (-Li)

Table 3.93 lists and describes the output options for RS08 Assembler.

Table 3.93 Tool Settings — RS08 Assembler > Output Options

Option	Description
Object File Format (-F)	Defines the format for the output file generated by the Assembler.
Show label statistics (-Li)	Using the -Ll option, the Compiler appends statistical information about the compilation session to the specified file. The information includes Compiler options, code size (in bytes), stack usage (in bytes) and compilation time (in seconds) for each procedure of the compiled file. The Compiler appends the information to the specified filename (or the file make.txt, if no argument given). Set the TEXTPATH: Text File Path environment variable to store the file into the path specified by the environment variable. Otherwise the Compiler stores the file in the current directory.

Table 3.93 Tool Settings — RS08 Assembler > Output Options (*continued*)

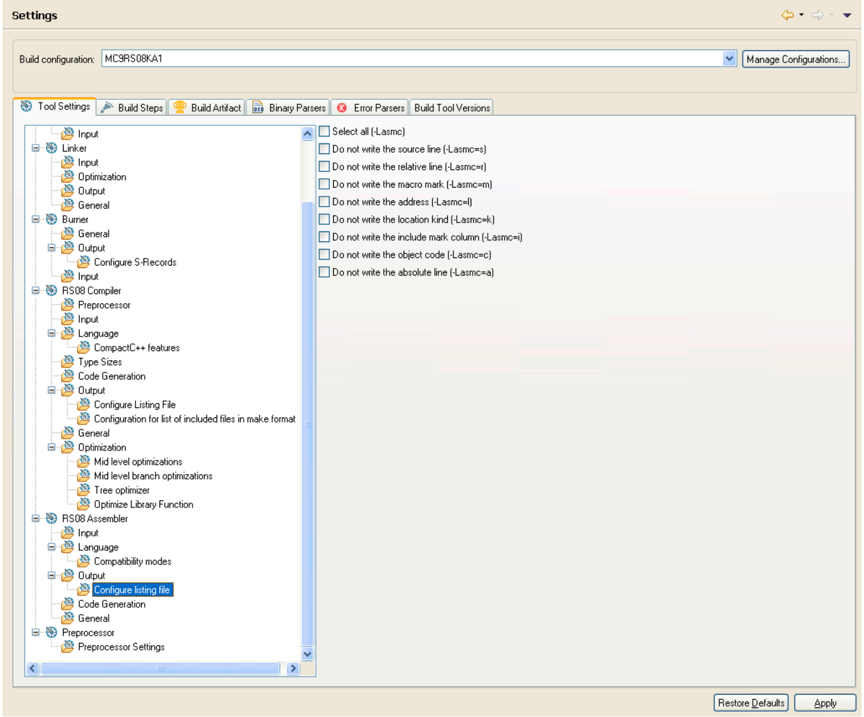
Option	Description
Generate listing file (e.g. %(TEXTPATH)/ %n.lst) (-L)	The -Lasm option causes the Compiler to generate an assembler listing file directly. The Compiler also prints all assembler-generated instructions to this file. The option specifies the name of the file. If no name is specified, the Compiler takes a default of %n.lst. If the resulting filename contains no path information the Compiler uses the TEXTPATH: Text File Path environment variable. The syntax does not always conform with the inline assembler or the assembler syntax. Therefore, use this option only to review the generated code. It cannot currently be used to generate a file for assembly.
Address size in the listing file (-Lasms)	Specifies the size of the addresses displayed in the listing. Options are: • 1 to display addresses as xx • 2 to display addresses as xxxx • 3 to display addresses as xxxxxx • 4 to display addresses asf xxxxxxxx
Do not print macro call in listing file (-Lc)	Specifies whether macro calls encountered in the source code are expanded and appear in the listing file.
Do not print macro definition in listing file (-Ld)	Instructs the Assembler to generate a listing file but not including any macro definitions. The listing file contains macro invocation and expansion lines as well as expanded include files.
Do not print macro expansion in listing file (-Le)	Switches on the generation of the listing file, but macro expansions are not present in the listing file. The listing file contains macro definition and invocation lines as well as expanded include files.
Do not print included files in listing file (-Li)	Switches on the generation of the listing file, but include files are not expanded in the listing file. The listing file contains macro definition, invocation, and expansion lines.

RS08 Assembler > Output > Configure Listing File

Use this panel to configure the listing file options of RS08 assembler.

Figure 3.100 shows the **Configure Listing File** panel.

Figure 3.100 Tool Settings — RS08 Assembler > Output > Configure Listing File


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Select all (-Lasmc)
☐ Do not write the source line (-Lasmc=s)
☐ Do not write the relative line (-Lasmc=r)
☐ Do not write the macro mark (-Lasmc=m)
☐ Do not write the address (-Lasmc=l)
☐ Do not write the location kind (-Lasmc=k)
☐ Do not write the include mark column (-Lasmc=i)
☐ Do not write the object code (-Lasmc=c)
☐ Do not write the absolute line (-Lasmc=a)

Table 3.94 lists and describes the Configure Listing File options for RS08 Assembler.

Table 3.94 Tool Settings — RS08 Assembler > Configure Listing File Options

Option	Description
Select all (-Lasmc)	Print all the columns in the listing file
Do not write the source line (-Lasmc=s)	Do not print source column in the listing file
Do not write the relative line (-Lasmc=r)	Do not print relative column (Rel.) in the listing file
Do not write the macro mark (-Lasmc=m)	Do not print macro mark column in the listing file

Table 3.94 Tool Settings — RS08 Assembler > Configure Listing File Options (*continued*)

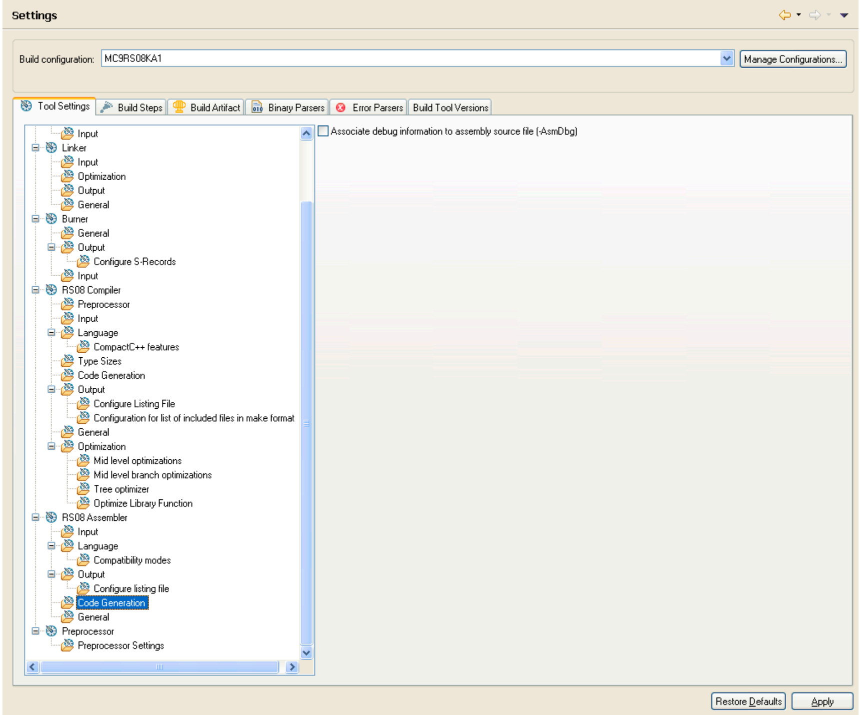
Option	Description
Do not write the address (-Lasmc=l)	Do not print address column (Loc) in the listing file
Do not write the location kind (-Lasmc=k)	Do not print the location type column in the listing file
Do not write the include mark column (-Lasmc=i)	Do not print the include mark column in the listing file
Do not write the object code (-Lasmc=c)	Do not print the object code in the listing file
Do not write the absolute line (-Lasmc=a)	Do not print the absolute column (Abs.) in the listing file

RS08 Assembler > Code Generation

Use this panel to specify the code generation options of the RS08 assembler.

Figure 3.101 shows the **Code Generation** panel.

Figure 3.101 Tool Settings — RS08 Assembler > Code Generation


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐ Associate debug information to assembly source file (-AsmDbg)

Table 3.95 lists and describes the Code Generation options for RS08 Assembler.

Table 3.95 Tool Settings — RS08 Assembler > Code Generation Options

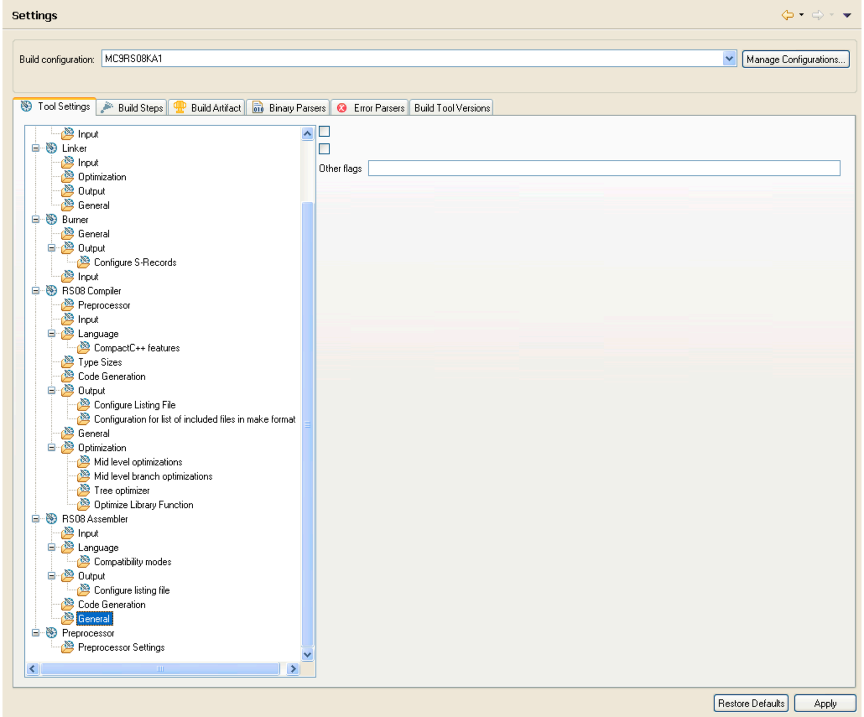
Option	Description
Associate debug information to assembly source file (-Asmdbg)	Passes the assembly source file name information to DWARF sections. When the output .abs file is debugged, the actual assembly source file is displayed instead of intermediary <filename>.dbg file.

RS08 Assembler > General

Use this panel to specify the general assembler behavior.

Figure 3.102 shows the **General** panel.

Figure 3.102 Tool Settings — RS08 Assembler > General


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MC9RS08KA1

☐

☐

Other flags

Table 3.96 lists and describes the general assembler options for RS08.

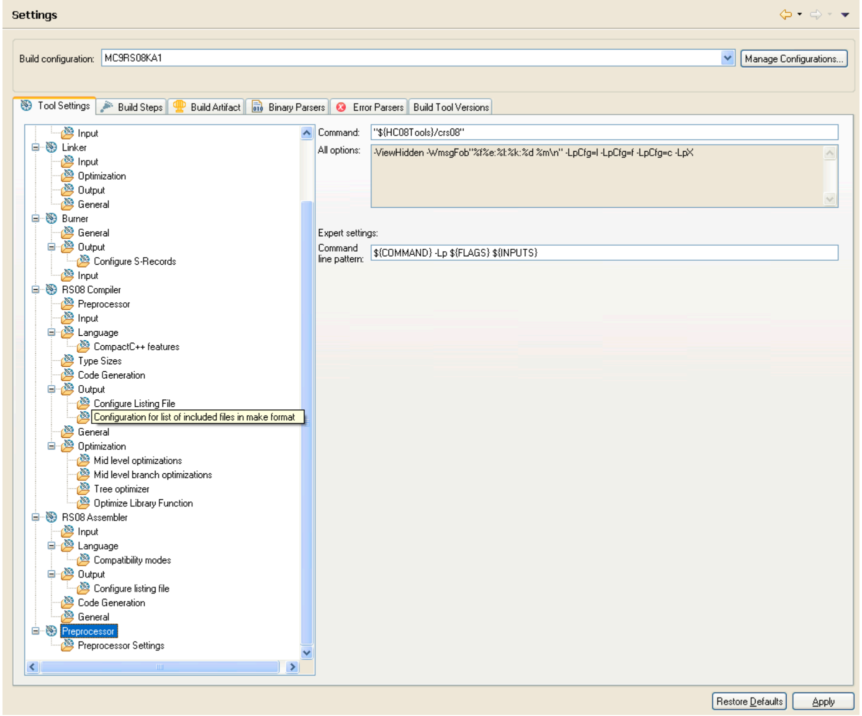
Table 3.96 Tool Settings — Assembler > General Options

Option	Description
MMU Support (-MMU)	Check to inform the compiler that CALL and RTC instructions are available, enabling code banking, and that the current architecture has extended data access capabilities, enabling support for `__linear` data types. This option can be used only when `-Cs08` is enabled.
MCUasm compatibility (-MCUasm)	Check to activate the compatibility mode with the MCUasm Assembler.
Other Flags	Specify additional command line options for the assembler; type in custom flags that are not otherwise available in the UI.

Preprocessor

Figure 3.103 shows the **Preprocessor** panel.

Figure 3.103 Tool Settings — Preprocessor


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Input
Linker
Input
Optimization
Output
General
Burner
General
Output
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Expert settings:
Restore Defaults
Apply

Build configuration: MC9RS08KA1

Command: "${HC08Tools}/crs08"

All options: -ViewHidden -WmsgFob"%f%e:%l:%k:%d %m\n" -LpCfg=l -LpCfg=f -LpCfg=c -LpX

Command line pattern: ${COMMAND} -Lp ${FLAGS} ${INPUTS}

Table 3.97 lists and describes the Preprocessor options for RS08.

Table 3.97 Tool Settings — Preprocessor Options

Option	Description
Command	Shows the location of the assembler executable file. Default value is: `"${HC08Tools}/crs08"`
All options	Shows the actual command line the assembler will be called with.

Table 3.97 Tool Settings — Preprocessor Options (*continued*)

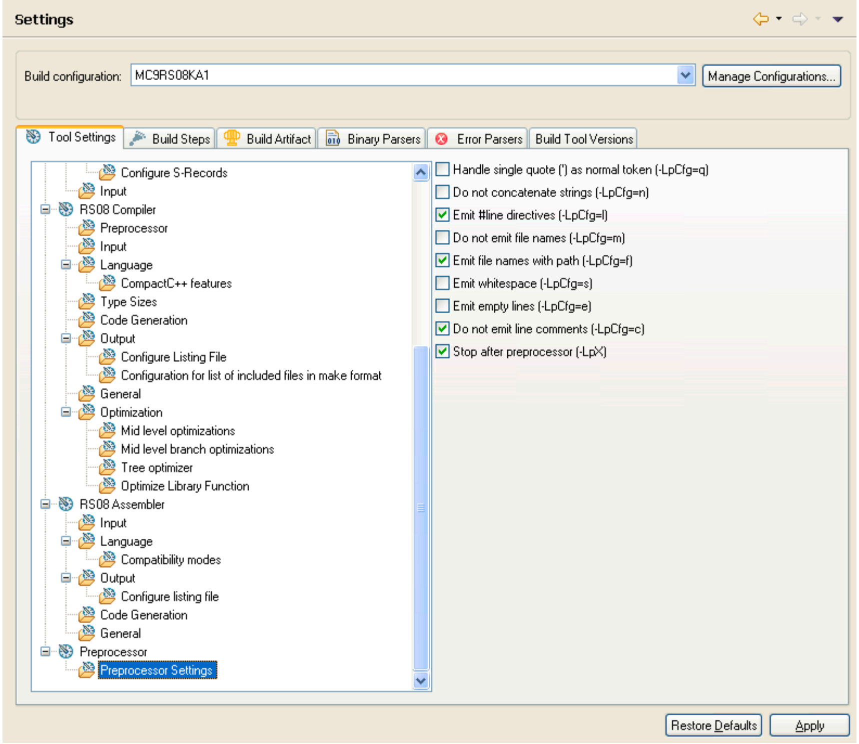
Option	Description
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} -Lp ${FLAGS} ${INPUTS}`.

Preprocessor > Preprocessor Settings

Use this panel to configure the preprocessor settings of the preprocessor.

Figure 3.104 shows the **Preprocessor Settings** panel.

Figure 3.104 Tool Settings — Preprocessor > Preprocessor Settings



Build configuration: MC9RS08KA1

☐ Handle single quote (') as normal token (-LpCfg=q)
☐ Do not concatenate strings (-LpCfg=n)
☑ Emit #line directives (-LpCfg=l)
☐ Do not emit file names (-LpCfg=m)
☑ Emit file names with path (-LpCfg=f)
☐ Emit whitespace (-LpCfg=s)
☐ Emit empty lines (-LpCfg=e)
☑ Do not emit line comments (-LpCfg=c)
☑ Stop after preprocessor (-LpX)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Configure S-Records
Input
RS08 Compiler
Preprocessor
Input
Language
CompactC++ features
Type Sizes
Code Generation
Output
Configure Listing File
Configuration for list of included files in make format
General
Optimization
Mid level optimizations
Mid level branch optimizations
Tree optimizer
Optimize Library Function
RS08 Assembler
Input
Language
Compatibility modes
Output
Configure listing file
Code Generation
General
Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.98 lists and describes the Preprocessor Settings options for RS08.

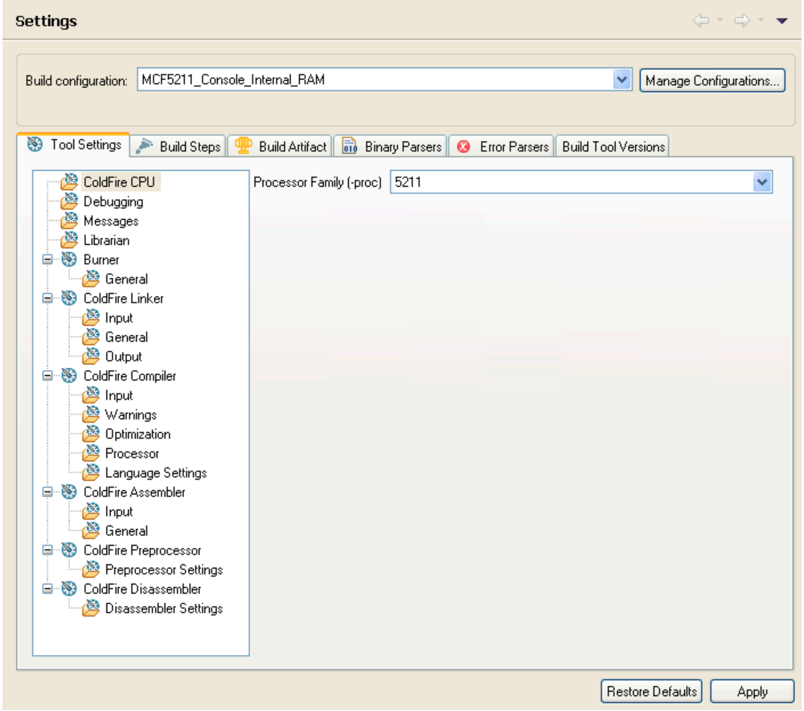
Table 3.98 Tool Settings — Assembler Options

Option	Description
Handle single quote (‘) as normal token (-LpCfg=q)	Use this option to handle single quote (‘) as normal token.
Do not concatenate strings (-LpCfg=n)	Use this option to avoid string concatenation.
Emit #line directive (-LpCfg=l)	Use this option to emit #line directives in preprocessor output.
Do not emit file names (-LpCfg=m)	Do not emit file names.
Emit file names with path (-LpCfg=f)	Use this option to emit file names with path.
Emit whitespace (-LpCfg=s)	Use this option to reconstruct spaces.
Emit empty lines(-LpCfg=e)	Use this option to emit empty lines.
Do not emit line comments (-LpCfg=c)	Do not emit line comments
Stop after preprocessor (-LpX)	Without this option, the compiler always translates the preprocessor output as C code. To do only preprocessing, use this option together with the -Lp option. No object file is generated.

Build Properties for ColdFire

The **Properties for *<project>*** window shows the corresponding build properties for a ColdFire project (Figure 3.105).

Figure 3.105 Build Properties - ColdFire Debug


Build configuration: MCF5211_Console_Internal_RAM

Processor Family (-proc) 5211

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Restore Defaults
Apply

Table 3.99 lists the build properties specific to developing software for ColdFire Debug. The properties that you specify in these panels apply to the selected build tool on the **Tool Settings** page of the **Properties for** ***<project>*** window.

Table 3.99 Build Properties for ColdFire Debug

Build Tool	Build Properties Panels
ColdFire CPU	ColdFire CPU
Debugging	Debugging
Messages	Messages
Librarian	Librarian
Burner	Burner > General

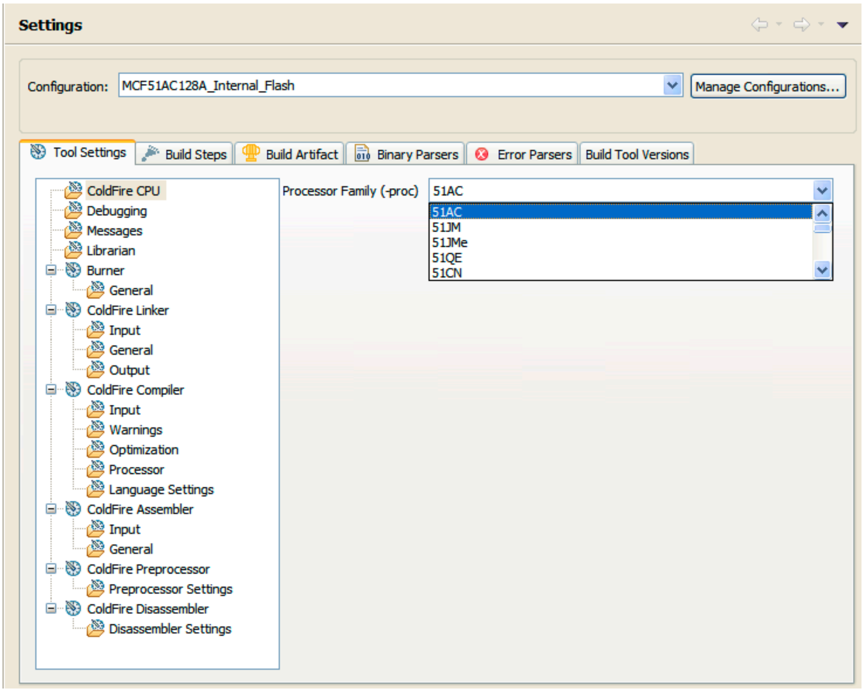
Table 3.99 Build Properties for ColdFire Debug (*continued*)

Build Tool	Build Properties Panels
ColdFire Linker	ColdFire Linker > Input
	ColdFire Linker > General
	ColdFire Linker > Output
ColdFire Compiler	ColdFire Compiler > Input
	ColdFire Compiler > Warnings
	ColdFire Compiler > Optimization
	ColdFire Compiler > Processor
	ColdFire Compiler > Language Settings
ColdFire Assembler	ColdFire Assembler > Input
	ColdFire Assembler > General
ColdFire Preprocessor	ColdFire Preprocessor > Preprocessor Settings
ColdFire Disassembler	ColdFire Disassembler > Disassembler Settings

ColdFire CPU

Use this panel to specify the CPU type, and the memory model that the architecture uses. The build tools (compiler, linker, and assembler) then use the properties set in this panel to generate CPU-specific code.

Figure 3.106 Tool Settings — ColdFire CPU


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
51AC
51JM
51JMe
51QE
51CN

Configuration: MCF51AC128A_Internal_Flash

Processor Family (-proc) 51AC

Table 3.100 lists and describes the ColdFire CPU options.

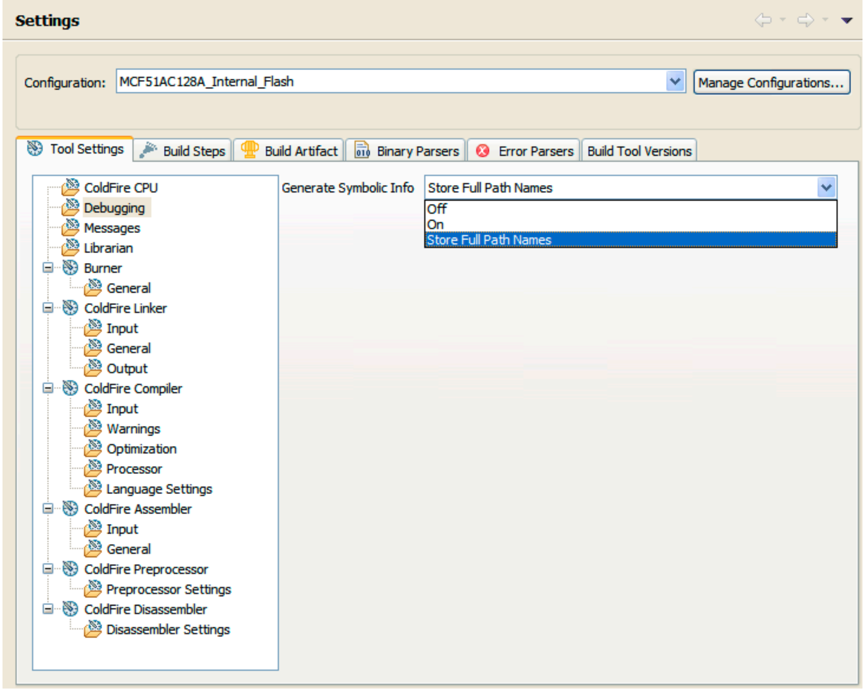
Table 3.100 Tool Settings — ColdFire CPU Options

Option	Description
Processor Family (-proc)	Lists the processor families supported by the ColdFire compiler. When you select a processor from this list, the compiler generates code that makes use of any of its hardware features or special instructions. For more detailed information on the features of each processor, refer to its reference manual document.

Debugging

Use this panel to specify the whether to generate symbolic information for debugging the build target (Figure 3.107).

Figure 3.107 Tool Settings — Debugging



Configuration: MCF51AC128A_Internal_Flash

Generate Symbolic Info: Store Full Path Names

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Off
On
Store Full Path Names

Table 3.101 lists and describes the debugging options.

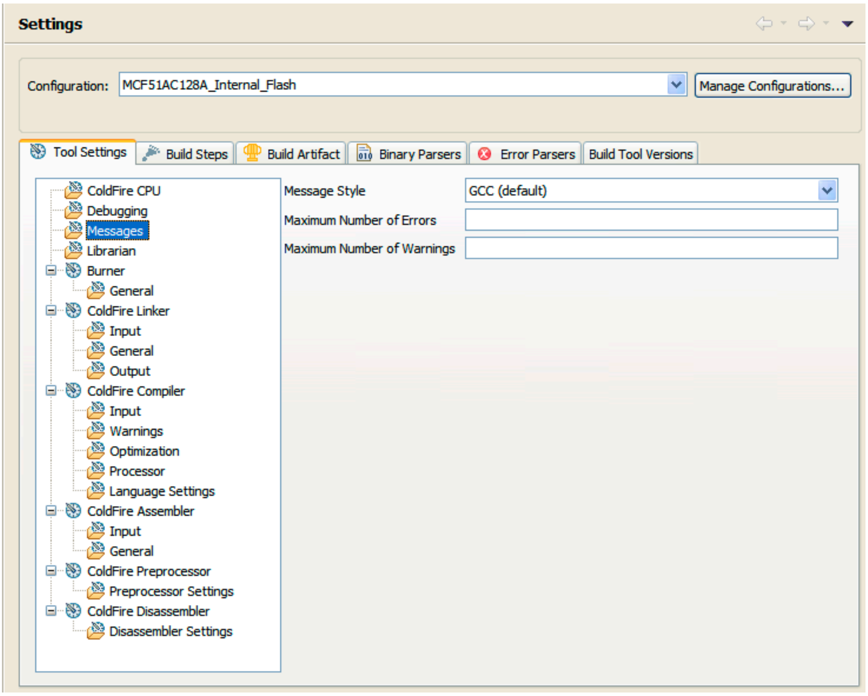
Table 3.101 Tool Settings — Debugging Options

Option	Description
Generate Symbolic Info	Specify whether to generate symbolic information for debugging: • **Off -** Select if you do not want to generate symbolic information for debugging the build target. • **On -** Select to generate symbolic information for debugging the build target. • **Store Full Path Names -** Select to generate symbolic information and store full path names for debugging the build target.

Messages

Use this panel to specify the whether to generate symbolic information for debugging the build target (Figure 3.108).

Figure 3.108 Tool Settings — Messages



Settings

Configuration: MCF51AC128A_Internal_Flash

Message Style: GCC (default)

Maximum Number of Errors:

Maximum Number of Warnings:

Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Table 3.102 lists and describes the message options.

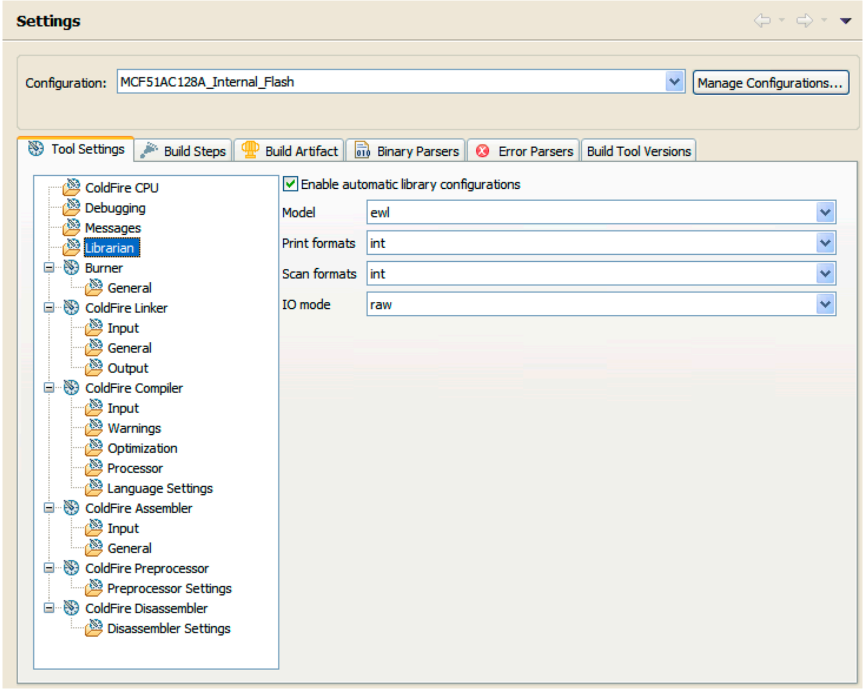
Table 3.102 Tool Settings — Messages Options

Option	Description
Message Style	List options to select message style. • **GCC(default)** — Uses the message style of the Gnu Compiler Collection tools • **MPW** — Uses the Macintosh Programmer's Workshop (MPW®) message style • **Standard** — Uses the standard message style • **IDE** — Uses context-free machine parseable message style • **Enterprise-IDE —** Uses CodeWarrior's Integrated Development Environment (IDE) message style. • **Parseable** — Uses parseable message style.
Maximum Number of Errors	Specify the number of errors allowed until the application stops processing.
Maximum Number of Warnings	Specify the maximum number of warnings.

Librarian

Use this panel to select whether the linker will identify standard libraries (Figure 3.109).

Figure 3.109 Tool Settings — Librarian


Settings
Configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

☑ Enable automatic library configurations

Model: ewl

Print formats: int

Scan formats: int

IO mode: raw

Table 3.103 lists and describes the librarian options.

Table 3.103 Tool Settings — Librarian Options

Option	Description
Enable automatic library configurations	Select to let the compiler identify standard libraries.
Model	Select a standard complying or EWL model from the drop-down list. EWL lets you precisely define the I/O operations. EWL drastically reduces the size of executables as you explicitly select the appropriate I/O behavior. Options are: `ewl`, `c9x`, `ewl_c++`, and `c9x_c++`.

Table 3.103 Tool Settings — Librarian Options (*continued*)

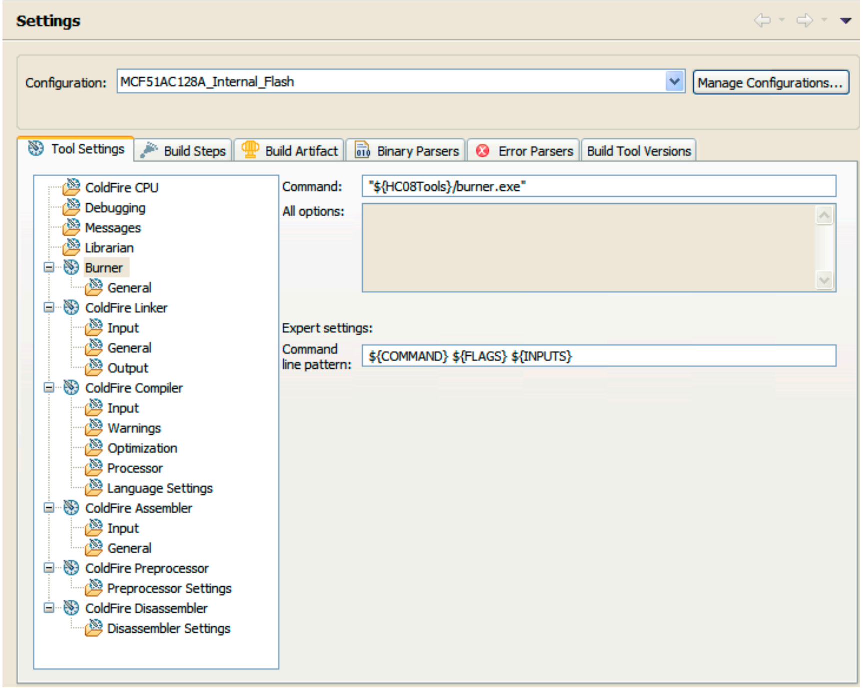
Option	Description
Print formats	Select the print formats from the drop-down list. The available options are: `int`, `int_FP`, `int_LL`, and `int_LL_FP`.
Scan formats	Select the scan formats from the drop-down list. The available options are: `int`, `int_FP`, `int_LL`, and `int_LL_FP`.
IO Mode	Select the input-output mode from the drop-down list. The available options are: `raw` and `buffered`.

Burner

Use the Burner for ColdFire Preference Panel to map *.bbl (batch burner language) files to the Burner Plug-In. When the project folder contains a *.bbl file, *.bbl file processing during the post-link phase uses the settings in the Burner preference panel.

Figure 3.110 shows the ColdFire Burner settings.

Figure 3.110 Tool Settings > Burner



Configuration: MCF51AC128A_Internal_Flash

Command: "${HC08Tools}/burner.exe"

All options:

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${INPUTS}

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Table 3.104 lists and describes the burner options for ColdFire.

Table 3.104 Tool Settings — Burner Options

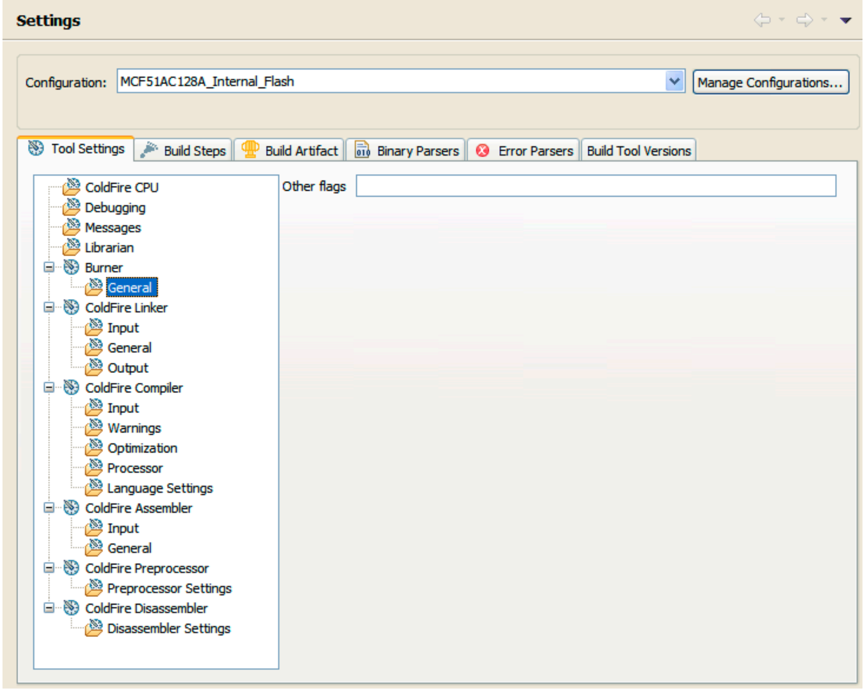
Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${HC08Tools}/burner.exe"`
All options	Shows the actual command line the burner will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${INPUTS}`.

Burner > General

Use this panel to specify other flags for the ColdFire Burner to use.

Figure 3.111 shows the **General** panel.

Figure 3.111 Tool Settings — Burner > General



Settings
Configuration:
MCF51AC128A_Internal_Flash
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Other flags

Table 3.105 lists and describes the general options for ColdFire burner.

Table 3.105 Tool Settings — Burner > General Options

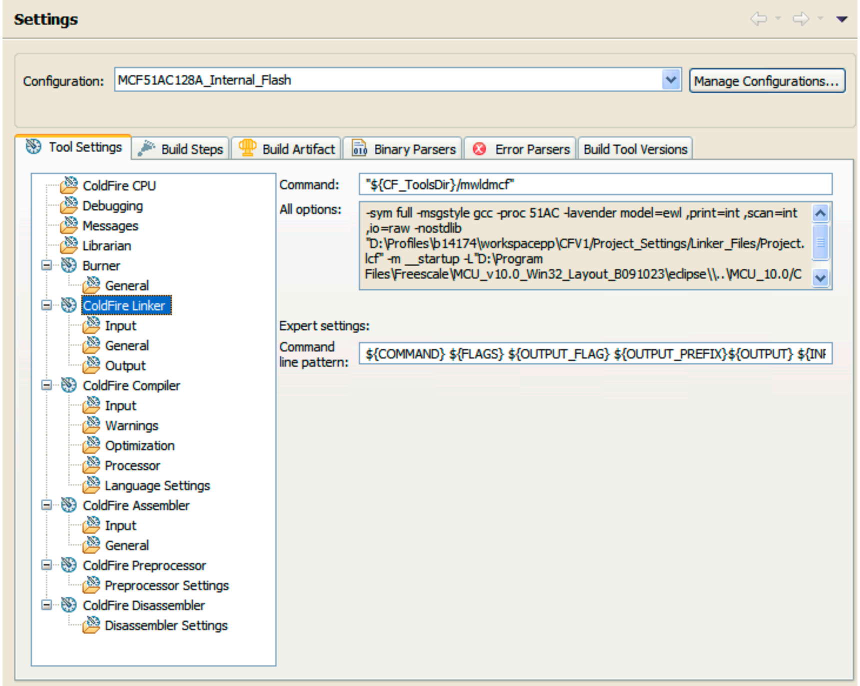
Option	Description
Other flags	Specify additional command line options for the burner; type in custom flags that are not otherwise available in the UI.

ColdFire Linker

Use this panel to specify ColdFire linker behavior. You can specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the input, general, and output settings.

Figure 3.112 shows the **ColdFire Linker** settings.

Figure 3.112 Tool Settings — ColdFire Linker


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Command: "${CF_ToolsDir}/mwldmcf"

All options:

```
-sym full -msgstyle gcc -proc 51AC -lavender model=ewl ,print=int ,scan=int
,io=raw -nostdlib
"D:\Profiles\b14174\workspacepp\CFV1/Project_Settings/Linker_Files/Project.
lcf" -m __startup -L"D:\Program
Files\Freescale\MCU_v10.0_Win32_Layout_B091023\eclipse\\..\MCU_10.0/C
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INI

Table 3.106 lists and describes the linker options for ColdFire.

Table 3.106 Tool Settings — ColdFire Linker Options

Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${CF_ToolsDir}/mwldmcf"`.
All options	Shows the actual command line the ColdFire linker will be called with.

Table 3.106 Tool Settings — ColdFire Linker Options (*continued*)

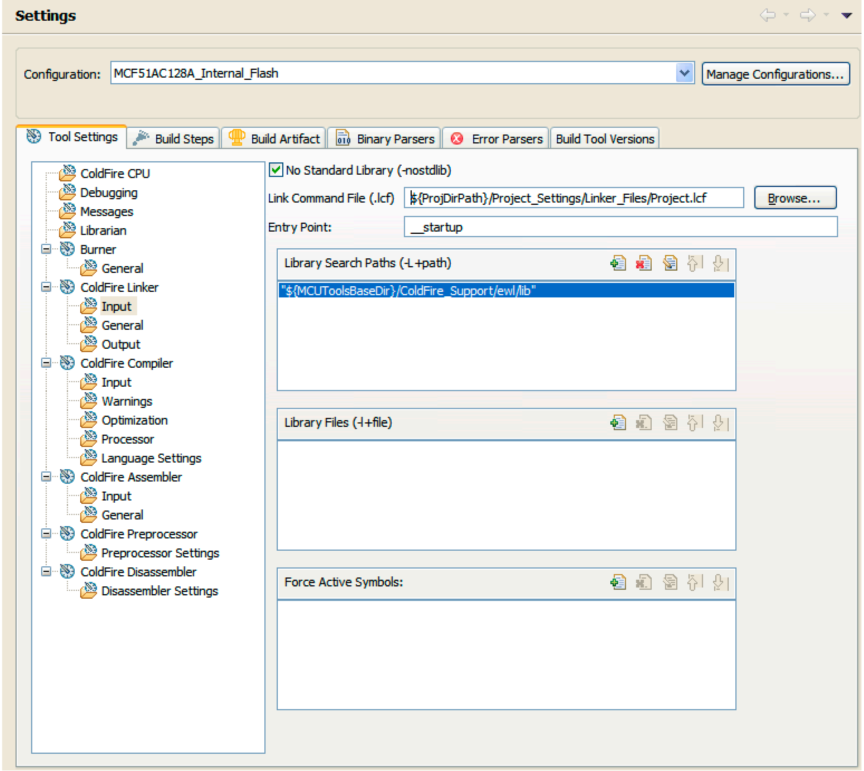
Option	Description
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

ColdFire Linker > Input

Use this panel to specify files the **ColdFire Linker** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.113 shows the **Input** panel.

Figure 3.113 Tool Settings — ColdFire Linker > Input


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Browse...

Configuration: MCF51AC128A_Internal_Flash

☑ No Standard Library (-nostdlib)

Link Command File (.lcf) ${ProjDirPath}/Project_Settings/Linker_Files/Project.lcf

Entry Point: __startup

Library Search Paths (-L+path)

"${MCUToolsBaseDir}/ColdFire_Support/ewl/lib"

Library Files (-l+file)

Force Active Symbols:

Table 3.107 lists and describes the input options for ColdFire.

Table 3.107 Tool Settings — ColdFire Linker > Input Options

Option	Description
No Standard Library (-nostdlib)	Select if there is no standard library attached
Link Command File (.lcf)	Consists of three kinds of segments, which must be in this order: • A memory segment, which begins with the MEMORY{} directive • Optional closure segments, which begin with the FORCE_ACTIVE{}, KEEP_SECTION{}, or REF_INCLUDE{} directives • A sections segment, which begins with the SECTIONS{} directive
Entry Point	Specifies the program starting point: the first function the debugger uses upon program start; default: __start. This default function is in file ColdFire__startup.c. It sets up the ColdFire EABI environment before code execution. Its final task is calling main().
Library Search Paths (-L +path)	Specifies the search pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.
Library Files ?(-l +file)	Specifies the pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.
Force Active Symbols	Disables deadstripping for particular symbols, enter the symbol names in the Force Active Symbols text box of the ColdFire Linker Panel.

ColdFire Linker > General

Use this panel to specify the general linker behavior.

Figure 3.114 shows the **General** panel.

Figure 3.114 Tool Settings — ColdFire Linker > General

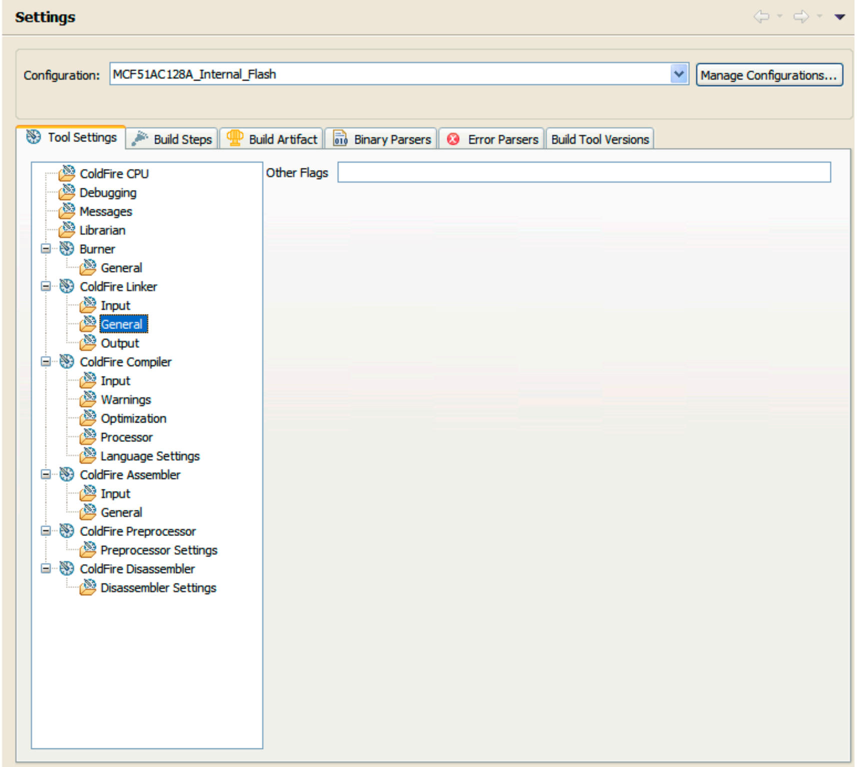

Settings
Configuration: MCF51AC128A_Internal_Flash
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Other Flags

Table 3.108 lists and describes the general linker options for ColdFire.

Table 3.108 Tool Settings — ColdFire Linker > General Options

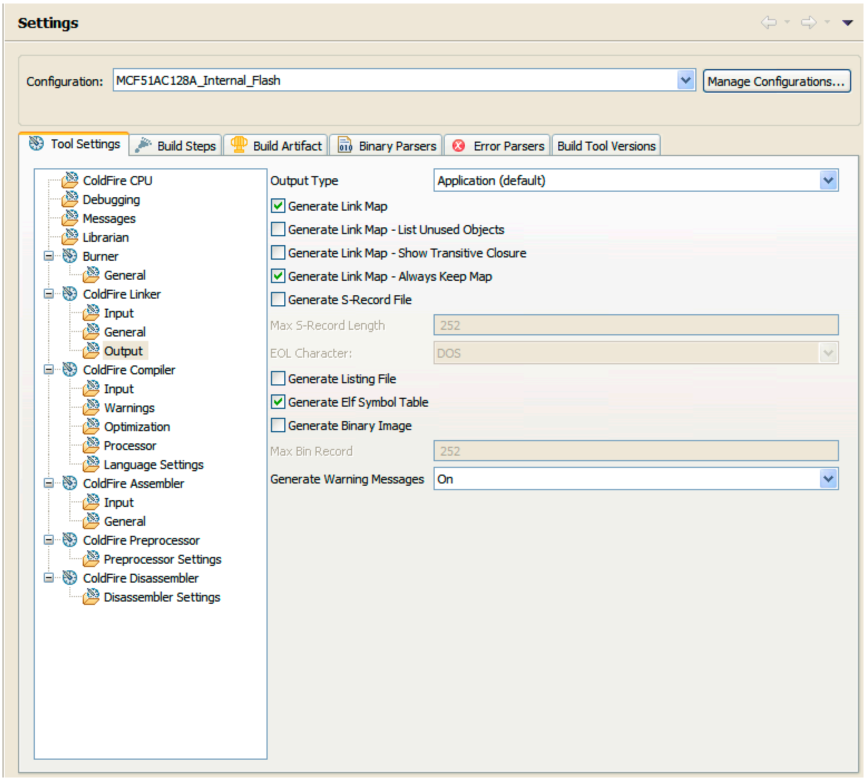
Option	Description
Other Flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

ColdFire Linker > Output

Use this panel to specify the output settings for the ColdFire linker.

Figure 3.115 shows the **Output** panel.

Figure 3.115 Tool Settings — ColdFire Linker > Output


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Output Type: Application (default)

☑ Generate Link Map

☐ Generate Link Map - List Unused Objects

☐ Generate Link Map - Show Transitive Closure

☑ Generate Link Map - Always Keep Map

☐ Generate S-Record File

Max S-Record Length: 252

EOL Character: DOS

☐ Generate Listing File

☑ Generate Elf Symbol Table

☐ Generate Binary Image

Max Bin Record: 252

Generate Warning Messages: On

Table 3.109 lists and describes the output settings for ColdFire linker.

Table 3.109 Tool Settings — ColdFire Linker > Output Options

Option	Description
Output Type	Select application as Application (default), Static Library, or Shared Library.
Generate Link Map	Check to generate link map.
Generate Link Map - List Unused Objects	Check to generate link map and list unused objects; appears grayed out if the **Generate Link Map** checkbox is not checked.

Table 3.109 Tool Settings — ColdFire Linker > Output Options (*continued*)

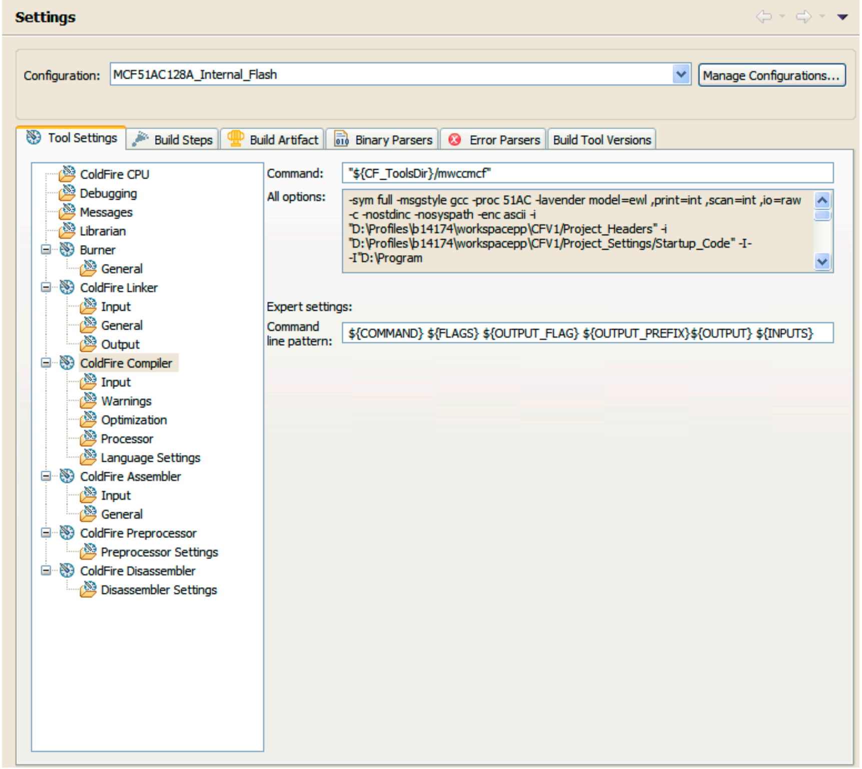
Option	Description
Generate Link Map -Show Transitive Closure	Check to generate link map and show transitive closure; appears grayed out if the **Generate Link Map** checkbox is not checked.
Generate Link Map -Always Keep Map	Check to generate link map and always keep the map; appears grayed out if the **Generate Link Map** checkbox is not checked.
Generate Link Map - Generate S-Record File	Check to generate link map and generate a S-record file.
Max S-Record Length	Specify the maximum length for S-record; appears grayed out if the **Generate S-Record File** checkbox is not checked. The default value is 252.
EOL Character	Specify the end-of-line character; appears grayed out if the **Generate S-Record File** checkbox is not checked. The default value is DOS.
Generate Listing File	Check to generate a listing file named lstfil.lst.
Generate Elf Symbol Table	Check to generate an ELF symbol table.
Generate Binary Image	Check to generate a binary image.
Max Bin Record	Specify the maximum value for bin record; appears grayed out if the **Generate Binary Image** checkbox is not checked. The default value is 252.
Generate Warning Messages	Select whether you want to generate warning messages, warn superseded definitions, or treat warnings as errors.

ColdFire Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the ColdFire Compiler tree control includes the general and the file search path settings.

Figure 3.116 shows the ColdFire Compiler settings.

Figure 3.116 Tool Settings — ColdFire Compiler



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Command: "${CF_ToolsDir}/mwccmcf"

All options:

```
-sym full -msgstyle gcc -proc 51AC -lavender model=ewl ,print=int ,scan=int ,io=raw
-c -nostdinc -nosyspath -enc ascii -i
"D:\Profiles\b14174\workspacepp\CFV1/Project_Headers" -i
"D:\Profiles\b14174\workspacepp\CFV1/Project_Settings/Startup_Code" -I-
-I"D:\Program
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.110 lists and describes the compiler options for ColdFire.

Table 3.110 Tool Settings — Compiler Options

Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${CF_ToolsDir}/mwccmcf"`.
All options	Shows the actual command line the ColdFire compiler will be called with.

Table 3.110 Tool Settings — Compiler Options (*continued*)

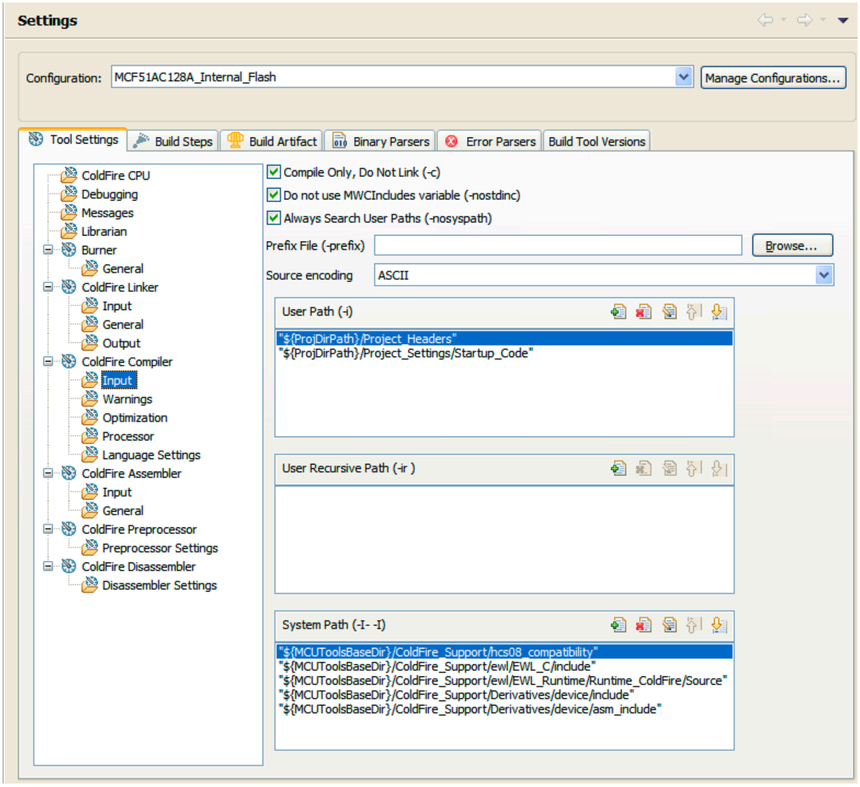
Option	Description
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

ColdFire Compiler > Input

Use this panel to specify additional files the **ColdFire Compiler** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.117 shows the **Input** panel.

Figure 3.117 Tool Settings — ColdFire Compiler > Input


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Browse...

Configuration: MCF51AC128A_Internal_Flash

☑ Compile Only, Do Not Link (-c)
☑ Do not use MWCIncludes variable (-nostdinc)
☑ Always Search User Paths (-nosyspath)

Prefix File (-prefix)

Source encoding: ASCII

User Path (-i)

```
"${ProjDirPath}/Project_Headers"
"${ProjDirPath}/Project_Settings/Startup_Code"
```

User Recursive Path (-ir)

System Path (-I- -I)

```
"${MCUToolsBaseDir}/ColdFire_Support/hcs08_compatibility"
"${MCUToolsBaseDir}/ColdFire_Support/ewl/EWL_C/include"
"${MCUToolsBaseDir}/ColdFire_Support/ewl/EWL_Runtime/Runtime_ColdFire/Source"
"${MCUToolsBaseDir}/ColdFire_Support/Derivatives/device/include"
"${MCUToolsBaseDir}/ColdFire_Support/Derivatives/device/asm_include"
```

Table 3.111 lists and describes the input options for ColdFire compiler.

Table 3.111 Tool Settings — ColdFire Compiler > Input Options

Option	Description
Compile only, Do Not Link (-c)	Check if you want to compile only and do not want to link the file.
Do not use MWCIncludes variable (-nostdinc)	Check if you do not want to use MWCIncludes variable.
Always Search User Paths (-nosyspath)	Check if you want to always search user paths.

Table 3.111 Tool Settings — ColdFire Compiler > Input Options (*continued*)

Option	Description
User Path (-i)	Lists the available user paths.
System Path	Lists the available system paths.

Table 3.112 lists and describes the toolbar buttons that help work with the user and system search paths.

Table 3.112 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.118) and specify the search path.
✖	**Delete** — Click to delete the selected search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.119) and update the selected search path.
⇧	**Move up** — Click to move the selected search path one position higher in the list
⇩	**Move down** — Click to move the selected search path one position lower in the list

Figure 3.118 shows the **Add directory path** dialog box.

Figure 3.118 Add directory path Dialog Box

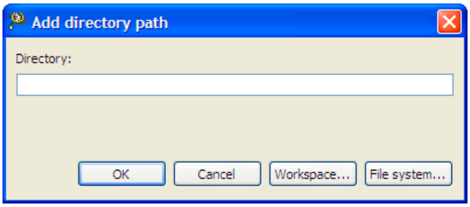
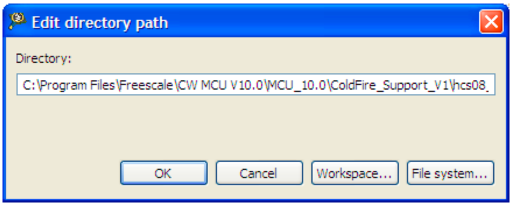


Figure 3.119 shows the **Edit directory path** dialog box.

Figure 3.119 Edit directory path Dialog Box



Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

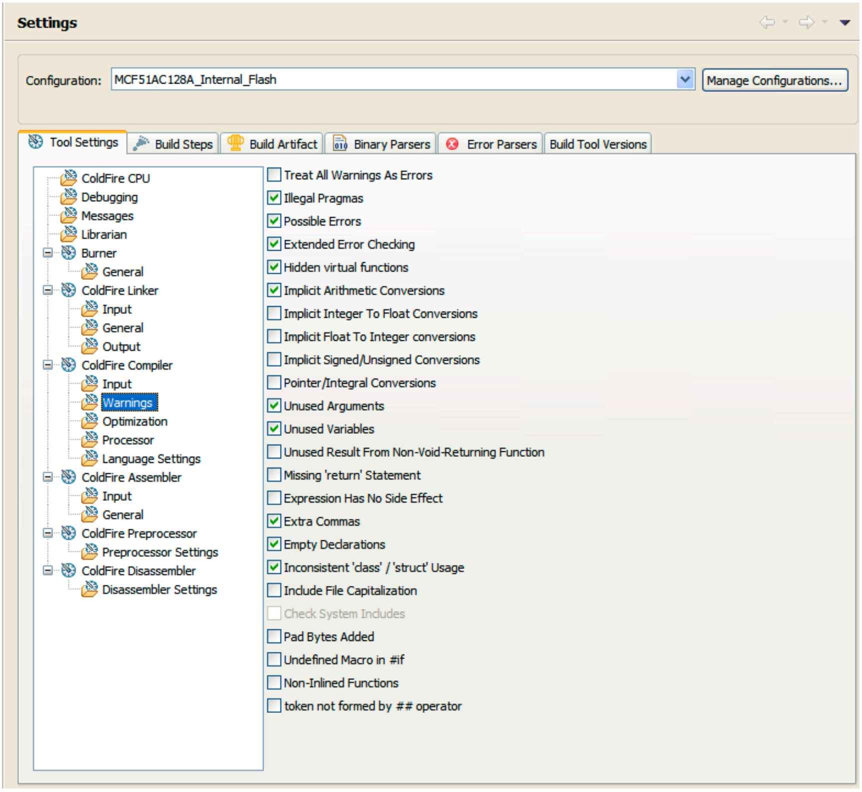
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

ColdFire Compiler > Warnings

Use this panel to control how the ColdFire compiler formats the listing file, as well as error and warning messages.

Figure 3.120 shows the **Warnings** panel.

Figure 3.120 Tool Settings — ColdFire Compiler > Warnings


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

☐ Treat All Warnings As Errors
☑ Illegal Pragmas
☑ Possible Errors
☑ Extended Error Checking
☑ Hidden virtual functions
☑ Implicit Arithmetic Conversions
☐ Implicit Integer To Float Conversions
☐ Implicit Float To Integer conversions
☐ Implicit Signed/Unsigned Conversions
☐ Pointer/Integral Conversions
☑ Unused Arguments
☑ Unused Variables
☐ Unused Result From Non-Void-Returning Function
☐ Missing 'return' Statement
☐ Expression Has No Side Effect
☑ Extra Commas
☑ Empty Declarations
☑ Inconsistent 'class' / 'struct' Usage
☐ Include File Capitalization
☐ Check System Includes
☐ Pad Bytes Added
☐ Undefined Macro in #if
☐ Non-Inlined Functions
☐ token not formed by ## operator

Table 3.113 lists and describes the warnings options for ColdFire compiler.

Table 3.113 Tool Settings — ColdFire Compiler > Warnings Options

Option	Description
Treat All Warnings As Errors	Check to treat all warnings as errors. The compiler will stop if it generates a warning message.
Illegal Pragmas	Check to notify the presence of illegal pragmas.
Possible Errors	Check to suggest possible errors.
Extended Error Checking	Check if you want to do an extended error checking.

Table 3.113 Tool Settings — ColdFire Compiler > Warnings Options (*continued*)

Option	Description
Hidden virtual functions	Check to generate a warning message if you declare a non-virtual member function that prevents a virtual function, that was defined in a superclass, from being called and is equivalent to pragma `warn_hidevirtual` and the command-line option `-warnings hidevirtual`.
Implicit Arithmentic Conversions	Check to warn of implict arithmetic conversions.
Implicit Integer to Float Conversions	Check to warn of implict conversion of an integer variable to floating-point type.
Implicit Float to Integer Conversions	Check to warn of implict conversions of a floating-point variable to integer type.
Implicit Signed/Unsigned Conversion	Check to enable warning of implict conversions between signed and unsigned variables.
Pointer/Integral Conversions	Check to enable warnings of conversions between pointer and integers.
Unused Arguments	Check to warn of unused arguments in a function.
Unused Variables	Check to warn of unused variables in the code.
Unused Result From Non-Void-Returning Function	Check to warn of unused result from non-void-returning functions.
Missing ‘return’ Statement	Check to warn of when a function lacks a return statement.
Expression Has No Side Effect	Check to issue a warning message if a source statement does not change the program's state. This is equivalent to the pragma `warn_no_side_effect`, and the command-line option `-warnings unusedexpr`.

Table 3.113 Tool Settings — ColdFire Compiler > Warnings Options (*continued*)

Option	Description
Extra Commas	Check to issue a warning message if a list in an enumeration terminates with a comma. The compiler ignores terminating commas in enumerations when compiling source code that conforms to the ISO/IEC 9899-1999 ("C99") standard and is equivalent to pragma `warn_extracomma` and the command-line `option -warnings extracomma.`
Empty Declarations	Check to warn of empty declarations.
Inconsistent ‘class’ / ‘struct’ Usage	Check to warn of inconsistent usage of class or struct.
Include File Capitalization	Check to issue a warning message if the name of the file specified in a #include "file" directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps` and the command-line option `-warnings filecaps.`
Check System Includes	Check to issue a warning message if the name of the file specified in a #include <file> directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps_system` and the command-line option `-warnings sysfilecaps.`
Pad Bytes Added	Check to issue a warning message when the compiler adjusts the alignment of components in a data structure and is equivalent to pragma `warn_padding` and the command-line option `-warnings padding.`
Undefined Macro in #if	Check to issues a warning message if an undefined macro appears in #if and #elif directives and is equivalent to pragma `warn_undefmacro` and the command-line option `-warnings undefmacro.`

Table 3.113 Tool Settings — ColdFire Compiler > Warnings Options (*continued*)

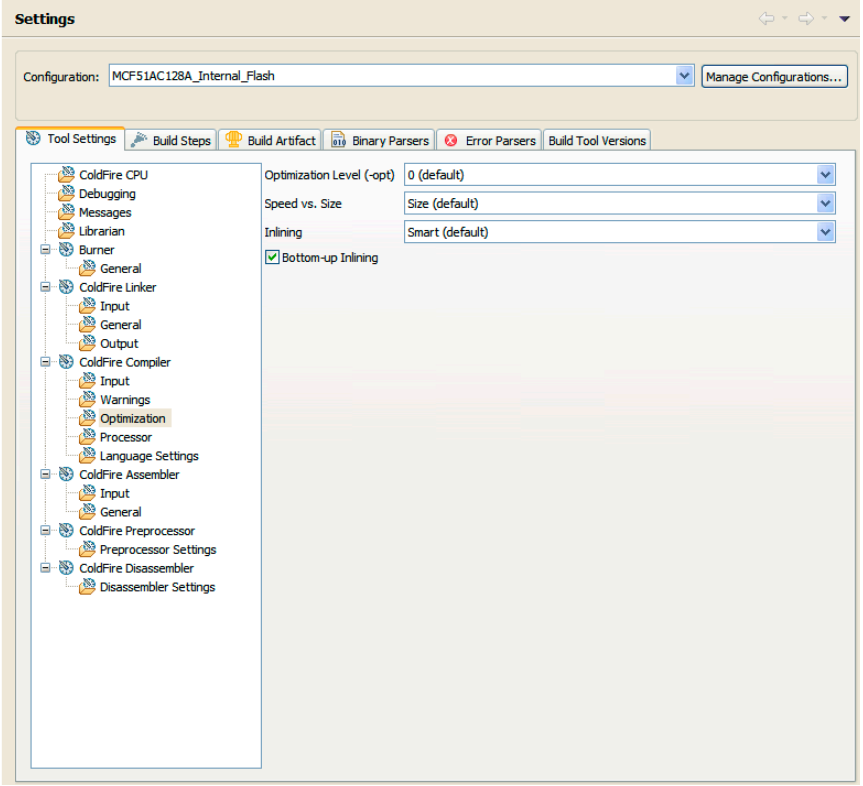
Option	Description
Non-Inlined Functions	Check to issue a warning message if a call to a function defined with the inline, __inline__, or __inline keywords could not be replaced with the function body and is equivalent to pragma `warn_notinlined` and the command-line option `-warnings notinlined`.
Token not formed by ## operator	Check to enable warnings for the illegal uses of the preprocessor's token concatenation operator (##). It is equivalent to the pragma `warn_illtokenpasting on`.

ColdFire Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.121 shows the **Optimization** panel.

Figure 3.121 Tool Settings — ColdFire Compiler > Optimization



Settings
Configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Optimization Level (-opt): 0 (default)

Speed vs. Size: Size (default)

Inlining: Smart (default)

☑ Bottom-up Inlining

Table 3.114 lists and defines each option of the **Optimization** panel.

Table 3.114 Tool Settings — ColdFire Compiler > Optimization Options

Option	Description
Optimization Level (-opt)	Specify the optimizations that you want the compiler to apply to the generated object code: • **0—**Disable optimizations. This setting is equivalent to specifying the `-O0` command-line option. The compiler generates unoptimized, linear assembly-language code. • **1—**The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the `-O1` command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • **2—**The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the `-O2` command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • **3—**The compiler performs all the level **2** optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the `-O3` command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level **2** optimizations.
Speed Vs Size	Use to specify an **Optimization Level** greater than **0**. • **Speed—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a faster execution speed, as opposed to a smaller executable code size. • **Size—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the `-Os` command-line option.

Table 3.114 Tool Settings — ColdFire Compiler > Optimization Options (*continued*)

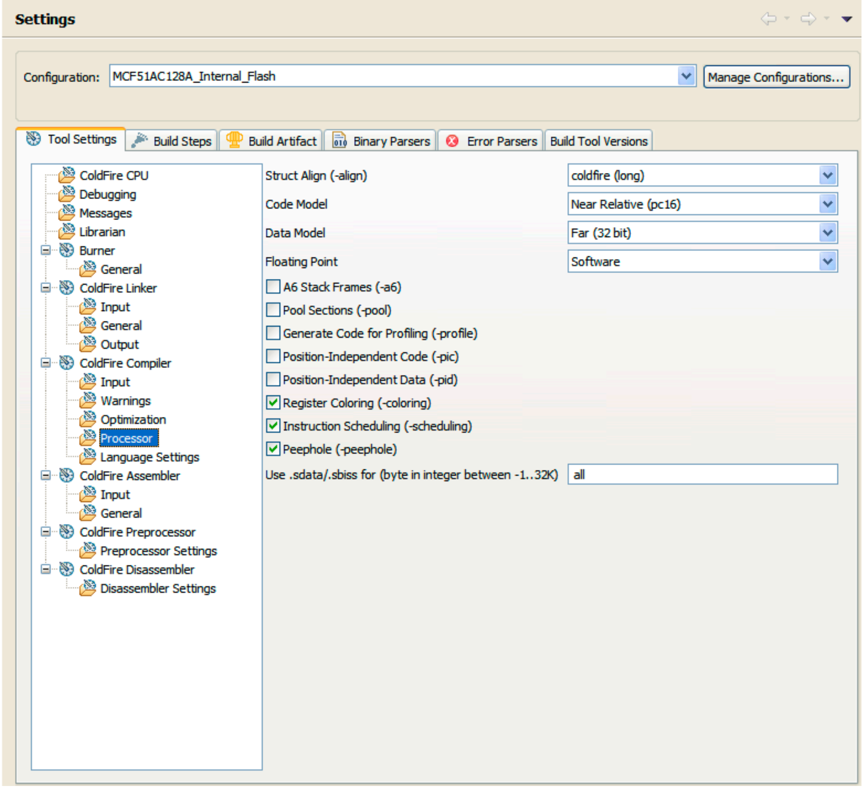
Option	Description
Inlining	Enables inline expansion. If there is a #pragma INLINE before a function definition, all calls of this function are replaced by the code of this function, if possible. Using the -Oi=c0 option switches off inlining. Functions marked with the #pragma INLINE are still inlined. To disable inlining, use the -Oi=OFF option.
Bottom-up Inlining	Check to control the bottom-up function inlining method. When active, the compiler inlines function code starting with the last function in the chain of functions calls, to the first one.

ColdFire Compiler > Processor

Use this panel to specify processor behavior. You can specify the file paths and define macros.

Figure 3.122 shows the **Processor** panel.

Figure 3.122 Tool Settings — ColdFire Compiler > Processor



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Struct Align (-align): coldfire (long)

Code Model: Near Relative (pc16)

Data Model: Far (32 bit)

Floating Point: Software

☐ A6 Stack Frames (-a6)
☐ Pool Sections (-pool)
☐ Generate Code for Profiling (-profile)
☐ Position-Independent Code (-pic)
☐ Position-Independent Data (-pid)
☑ Register Coloring (-coloring)
☑ Instruction Scheduling (-scheduling)
☑ Peephole (-peephole)

Use .sdata/.sbss for (byte in integer between -1..32K): all

Table 3.115 lists and defines each option of the **Processor** panel.

Table 3.115 Tool Settings — ColdFire Compiler > Processor Options

Option	Description
Struct Align (-align)	Specifies record and structure alignment in memory: • **Byte** — Aligns all fields on 1 byte boundaries • **68k (word)** — Aligns all fields on word boundaries • **coldfire (long)** — Aligns all fields on long word boundaries • **Default —** Coldfire (long). This panel element corresponds to the options align pragma. **Note**: When you compile and link, ensure that alignment is the same for all files and libraries.
Code Model	Specifies access addressing for data and instructions in the object code: • **Smart** — Relative (16-bit) for function calls in the same segment; otherwise absolute (32-bit) • **Near (16 bit)** — Relative for all function calls • **Far (32 bit)** — Absolute for all function calls
Data Model	Specifies global-data storage and reference: • **Far (32 bit)** — Storage in far data space; available memory is the only size limit. • **Near (16 bit)** — Storage in near data space; size limit is 64K. • **Default** — Far (32 bit). This panel element corresponds the far_data pragma

Table 3.115 Tool Settings — ColdFire Compiler > Processor Options (*continued*)

Option	Description
Floating Point	Specifies handling method for floating point operations: - **Software** — C runtime library code emulates floating-point operations. - **Hardware** — Processor hardware performs floating point operations; only appropriate for processors that have floating-point units. - **None** Default: Software For software selection, your project must include the appropriate FP_ColdFire C runtime library file. Grayed out if your target processor lacks an FPU.
A6 Stack Frame (-a6)	Clear to disable call-stack tracing; generates faster and smaller code. By default, the option is checked.
Pool Sections (-pool)	Check to collect all string constants into a single data object so your program needs one data section for all of them.
Generate Code for Profiling (-profile)	Check to enable the processor generate code for use with a profiling tool. Checking this box corresponds to using the command-line option `-profile`. Clearing this checkbox is equivalent to using the command-line option `-noprofile`
Position-Independent Code (-pic)	Check to generate position independent code (PIC) that is non relocatable.
Position-Independent Data (-pid)	Check to generate non-relocatable position-independent data (PID). PID is available with 16- and 32-bit addressing.
Register Coloring (-coloring)	Clear to enable the Compiler force all local variables to be stack-based except for compiler generated temporaries.

Table 3.115 Tool Settings — ColdFire Compiler > Processor Options (*continued*)

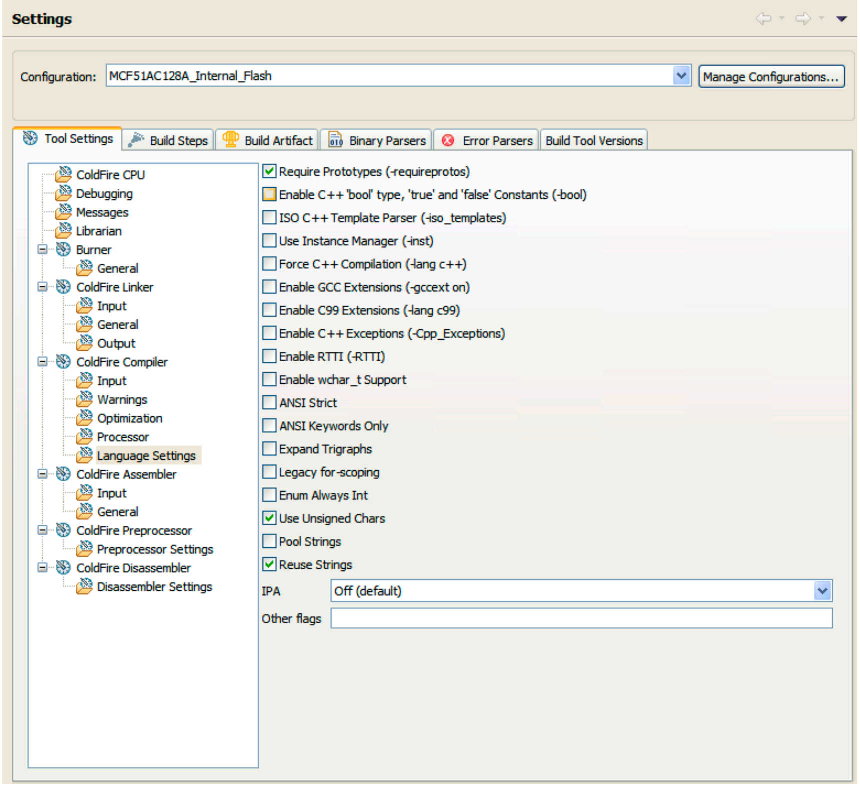
Option	Description
Instruction Scheduling (-scheduling)	Clear to prevent from scheduling instructions.
Peephole (-peephole)	Clear to prevent the compiler from compiling long instruction sequences into compact ones. By default, the option is checked. When on (default setting) it does not affect debugging unless the resulting instruction is a memory-to-memory operation which might make a variable used as temporary disappear.
Use .sdata.sbiss for (byte in integer between -1.32K)	The options are: • All data — Select this option button to store all data items in the small data address space • All data smaller than — Select this option button to specify the maximum size for items stored in the small data address space; enter the maximum size in the text box. Using the small data area speeds data access, but has ramifications for the hardware memory map. The default settings specify not using the small data area. By default, all data smaller than is checked.

ColdFire Compiler > Language Settings

Use this panel direct the ColdFire compiler to apply specific processing modes to the language source code. You can compile source files with just one collection at a time. To compile source files with multiple collections, you must compile the source code sequentially. After each compile iteration change the collection of settings that the ColdFire compiler uses.

Figure 3.123 shows the **Language Settings** panel.

Figure 3.123 Tool Settings — ColdFire Compiler > Language Settings


Settings
Configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

☑ Require Prototypes (-requireprotos)
☐ Enable C++ 'bool' type, 'true' and 'false' Constants (-bool)
☐ ISO C++ Template Parser (-iso_templates)
☐ Use Instance Manager (-inst)
☐ Force C++ Compilation (-lang c++)
☐ Enable GCC Extensions (-gccext on)
☐ Enable C99 Extensions (-lang c99)
☐ Enable C++ Exceptions (-Cpp_Exceptions)
☐ Enable RTTI (-RTTI)
☐ Enable wchar_t Support
☐ ANSI Strict
☐ ANSI Keywords Only
☐ Expand Trigraphs
☐ Legacy for-scoping
☐ Enum Always Int
☑ Use Unsigned Chars
☐ Pool Strings
☑ Reuse Strings

IPA: Off (default)

Other flags:

Table 3.116 lists and defines each option of the **Language Settings** panel.

Table 3.116 Tool Settings — ColdFire Compiler > Language Settings Options

Option	Description
Require Prototypes (-requireprotos)	Check to enforce the requirement of function prototypes. the compiler generates an error message if you define a previously referenced function that does not have a prototype. If you define the function before it is referenced but do not give it a prototype, this setting causes the compiler to issue a warning message.
Enable C++ ‘bool’ type, ‘true’ and ‘false’ Constants (-bool)	Check to enable the C++ compiler recognize the bool type and its true and false values specified in the ISO/IEC 14882-1998 C++ standard; is equivalent to pragma `bool` and the command-line option `-bool`.
ISO C++ Template Parser (-iso_templates)	Check to follow the ISO/IEC 14882-1998 standard for C++ to translate templates, enforcing more careful use of the typename and template keywords. The compiler also follows stricter rules for resolving names during declaration and instantiation and is equivalent to pragma `parse_func_templ` and the command-line `option -iso_templates`.
Use Instance Manager (-inst)	Check to reduce compile time by generating any instance of a C++ template (or non-inlined inline) function only once.
Force C++ Compilation (-lang c99)	Check to translates all C source files as C++ source code and is equivalent to pragma `cplusplus` and the command-line option `-lang c++`.
Enable GCC extensions (-gcc)	Check to recognize language features of the GNU Compiler Collection (GCC) C compiler that are supported by CodeWarrior compilers; is equivalent to pragma `gcc_extensions` and the command-line option `-gcc_extensions`.

Table 3.116 Tool Settings — ColdFire Compiler > Language Settings Options (*continued*)

Option	Description
Enable C99 Extensions (-lang c99)	Check to recognize ISO/IEC 9899-1999 ("C99") language features; is equivalent to pragma `c99` and the command-line option `-dialect c99`.
Enable C++ Exceptions (-Cpp_Exceptions)	Check to generate executable code for C++ exceptions; is equivalent to pragma `exceptions` and the command-line option `-cpp_exceptions`.
Enable RTTI (-RTTI)	Check to allow the use of the C++ runtime type information (RTTI) capabilities, including the `dynamic_cast` and `typeid` operators; is equivalent to pragma RTTI and the command-line option -RTTI.
Enable wchar_tSupport	Check to enable C++ compiler recognize the wchar_t data type specified in the ISO/IEC 14882-1998 C++ standard; is equivalent to pragma `wchar_type` and the command-line option `-wchar_t`.
ANSI Strict	Check to enable C compiler operate in strict ANSI mode. In this mode, the compiler strictly applies the rules of the ANSI/ISO specification to all input files. This setting is equivalent to specifying the `- ansi` command-line option. The compiler issues a warning for each ANSI/ISO extension it finds.
ANSI Keywords Only	Check to generate an error message for all non-standard keywords (ISO/IEC 9899-1990 C, §6.4.1). If you must write source code that strictly adheres to the ISO standard, enable this setting; is equivalent to pragma `only_std_keywords` and the command-line option `-stdkeywords`.
Expand Trigraphs	Check to recognize trigraph sequences (ISO/IEC 9899-1990 C, §5.2.1.1); is equivalent to **pragma** `trigraphs` **and the command-line option** `-trigraphs`.

Table 3.116 Tool Settings — ColdFire Compiler > Language Settings Options (*continued*)

Option	**Description**
Legacy for-scoping	Check to generate an error message when the compiler encounters a variable scope usage that the ISO/IEC 14882-1998 C++ standard disallows, but is allowed in the C++ language specified in The Annotated C++ Reference Manual ("ARM"); is equivalent to pragma `ARM_scoping` and the command-line option `-for_scoping`.
Enum Always Int	Check to use signed integers to represent enumerated constants and is equivalent to pragma `enumsalwaysint` and the command-line option `-enum`.
Use Unsigned Chars	Check to treat char declarations as unsigned char declarations and is equivalent to pragma `unsigned_char` and the command-line option `-char unsigned.`
Pool Strings	**Check to** collect all string constants into a single data section in the object code it generates and is equivalent to pragma `pool_strings` and the command-line `option -strings pool.`
Reuse Strings	**Check to** store only one copy of identical string literals and is equivalent to opposite of the pragma `dont_reuse_strings` and the command-line `option -string reuse.`

Table 3.116 Tool Settings — ColdFire Compiler > Language Settings Options (*continued*)

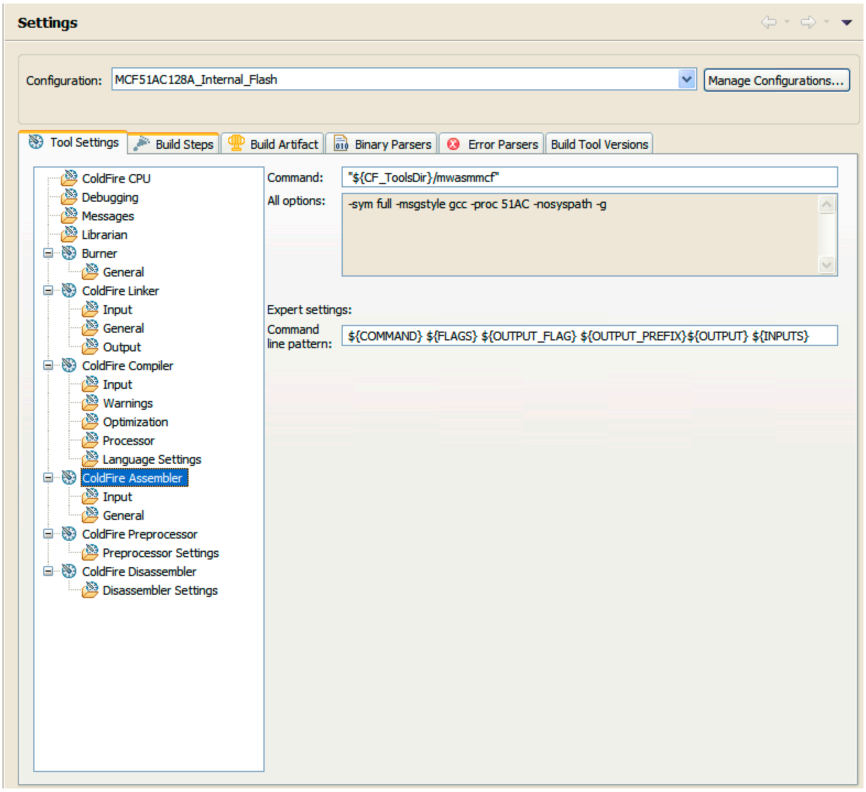
Option	Description
IPA	Specifies the Interprocedural Analysis (IPA) policy. • **Off** — No interprocedural analysis, but still performs function-level optimization. Equivalent to the "no deferred inlining" compilation policy of older compilers. • **File** — Completely parse each translation unit before generating any code or data. Equivalent to the "deferred inlining" option of older compilers. Also performs an early dead code and dead data analysis in this mode. Objects with unreferenced internal linkages will be dead-stripped in the compiler rather than in the linker. • **Program** —Completely parse the entire program before optimizing and generating code, providing many optimization benefits. For example, the compiler can auto-inline functions that are defined in another translation unit.
Other flags	Specify additional command line options for the compiler; type in custom flags that are not otherwise available in the UI. **Note**: To enable CodeWarrior MCU V10.x to generate .lst file for each source file in ColdFire, you need to specify -S in the **Other Flags** option.

ColdFire Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler. Additionally, the Assembler tree control includes the general and include file search path settings.

Figure 3.124 shows the **Assembler** settings.

Figure 3.124 Tool Settings — ColdFire Assembler


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
Expert settings:

Configuration: MCF51AC128A_Internal_Flash

Command: "${CF_ToolsDir}/mwasmmcf"

All options: -sym full -msgstyle gcc -proc 51AC -nosyspath -g

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.117 lists and defines each option of the **ColdFire Assembler** panel.

Table 3.117 Tool Settings — ColdFire Assembler Options

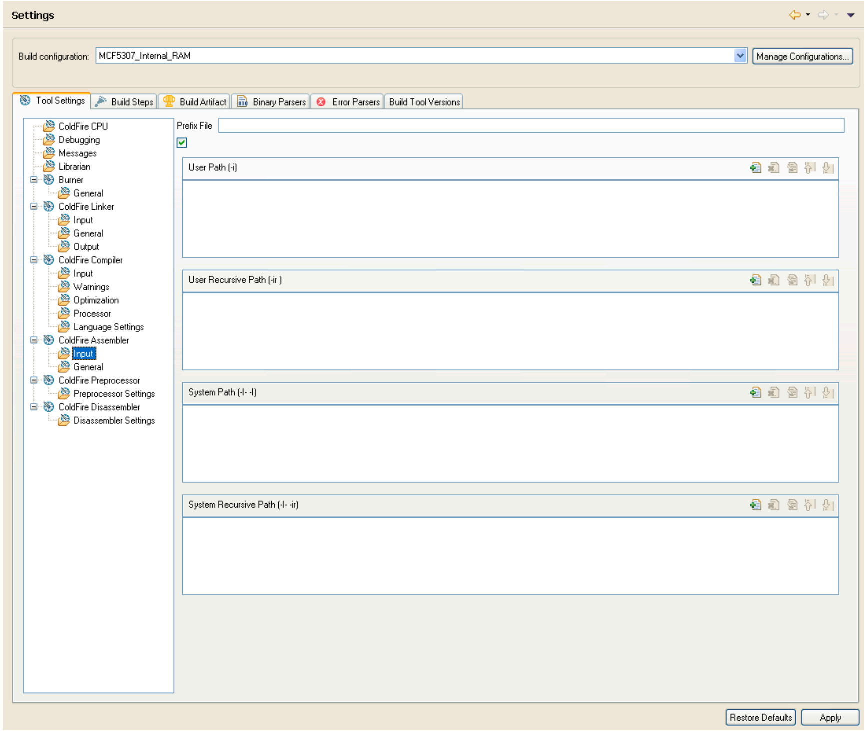
Option	Description
Command	Shows the location of the assembler executable file.
All options	Shows the actual command line the assembler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

ColdFire Assembler > Input

Use this panel to specify additional files the **ColdFire Assembler** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.125 shows the **Input** panel.

Figure 3.125 Tool Settings — ColdFire Assembler > Input



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings
User Path (-i)
User Recursive Path (-ir)
System Path (-I- -I)
System Recursive Path (-I- -ir)
Manage Configurations...
Restore Defaults
Apply

Build configuration: MCF5307_Internal_RAM

Prefix File

☑

Table 3.118 lists and describes the input options for ColdFire assembler.

Table 3.118 Tool Settings — ColdFire Assembler > Input Options

Option	Description
Prefix File	Specifies a file automatically included in all project assembly files.
User Path (-i)	Lists the available user paths.

Table 3.118 Tool Settings — ColdFire Assembler > Input Options (*continued*)

Option	Description
User Recursive Path (-ir)	
System Path (-I- -I)	Lists the available system paths.
System Recursive Path (-I- -ir)	

Table 3.119 lists and describes the toolbar buttons that help work with the user and system search paths.

Table 3.119 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.126) and specify the search path.
✖	**Delete** — Click to delete the selected search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.127) and update the selected search path.
⇧	**Move up** — Click to move the selected search path one position higher in the list
⇩	**Move down** — Click to move the selected search path one position lower in the list

Figure 3.126 shows the **Add directory path** dialog box.

Figure 3.126 Add directory path Dialog Box

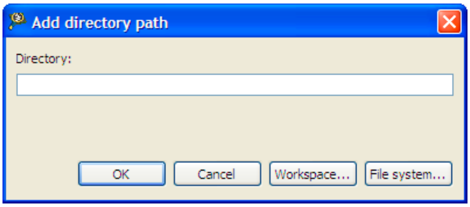
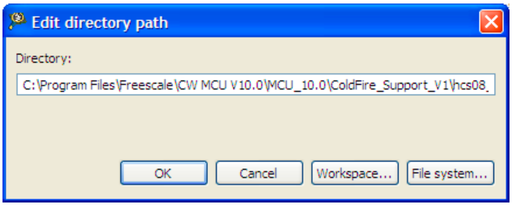

Add directory path
Directory:
OK
Cancel
Workspace...
File system...

Figure 3.127 shows the **Edit directory path** dialog box.

Figure 3.127 Edit directory path Dialog Box


Edit directory path
Directory:
OK
Cancel
Workspace...
File system...

Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

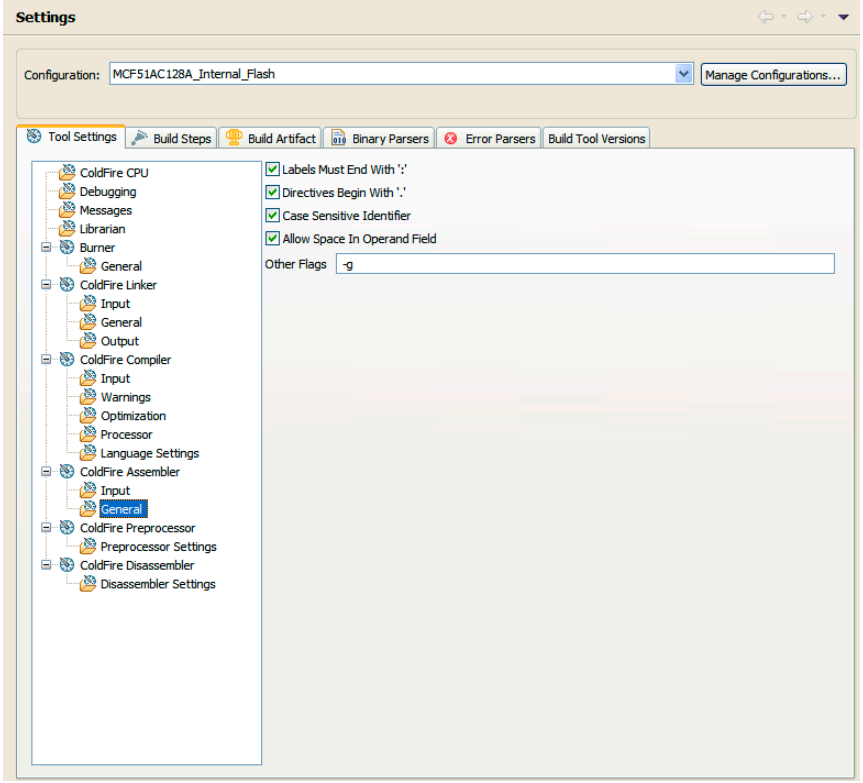
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

ColdFire Assembler > General

Use this panel to specify the general assembler behavior.

Figure 3.128 shows the **General** panel.

Figure 3.128 Tool Settings — ColdFire Assembler > General


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

☑ Labels Must End With ':'

☑ Directives Begin With '.'

☑ Case Sensitive Identifier

☑ Allow Space In Operand Field

Other Flags -g

Figure 3.120 lists and describes the general assembler options for ColdFire.

Table 3.120 Tool Settings — Assembler > General Options

Option	Description
Label Must End With ‘:’	Clear if system does not require labels to end with colons. By default, the option is checked.
Directives Begin With ‘.’	Clear if the system does not require directives to start with periods. By default, the option is checked.
Case Sensitive Identifier	Clear to instruct the assembler to ignore case in identifiers. By default, the option is checked.

Table 3.120 Tool Settings — Assembler > General Options (*continued*)

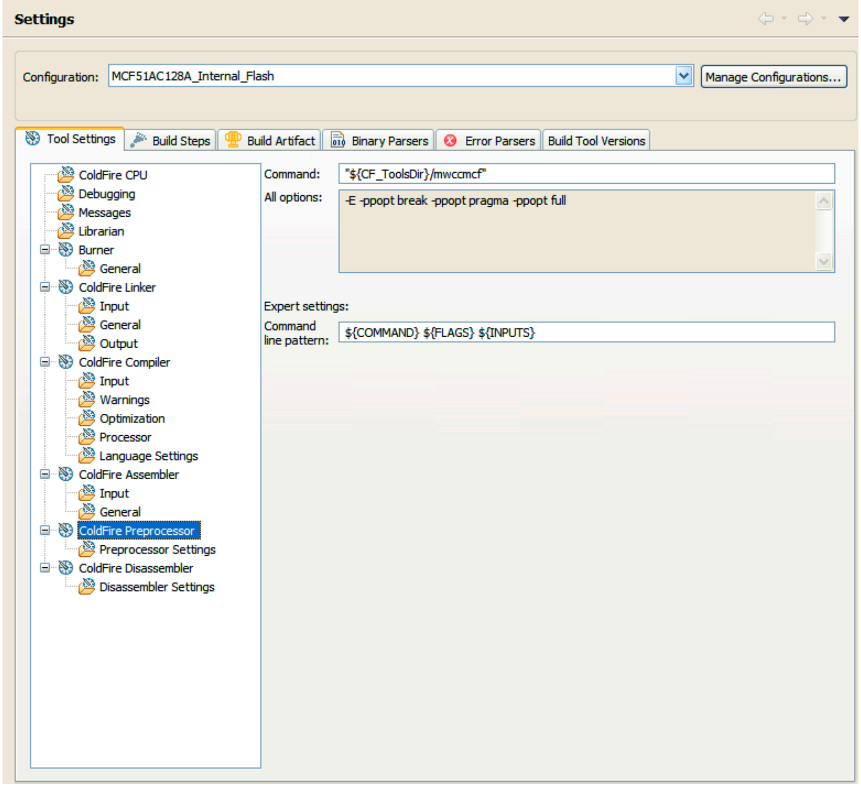
Option	Description
Allow Space In Operand Field	Clear to restrict the assembler from adding spaces in operand fields. By default, the option is checked.
Other Flags	Specify additional command line options for the assembler; type in custom flags that are not otherwise available in the UI.

ColdFire Preprocessor

Use this panel to specify preprocessor behavior and define macros.

Figure 3.129 shows the **Preprocessor** panel.

Figure 3.129 Tool Settings — ColdFire Preprocessor



Configuration: MCF51AC128A_Internal_Flash

Command: "${CF_ToolsDir}/mwccmcf"

All options: -E -ppopt break -ppopt pragma -ppopt full

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${INPUTS}

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Figure 3.67 lists and describes the preprocessor options for ColdFire.

Table 3.121 Tool Settings — ColdFire Preprocessor Options

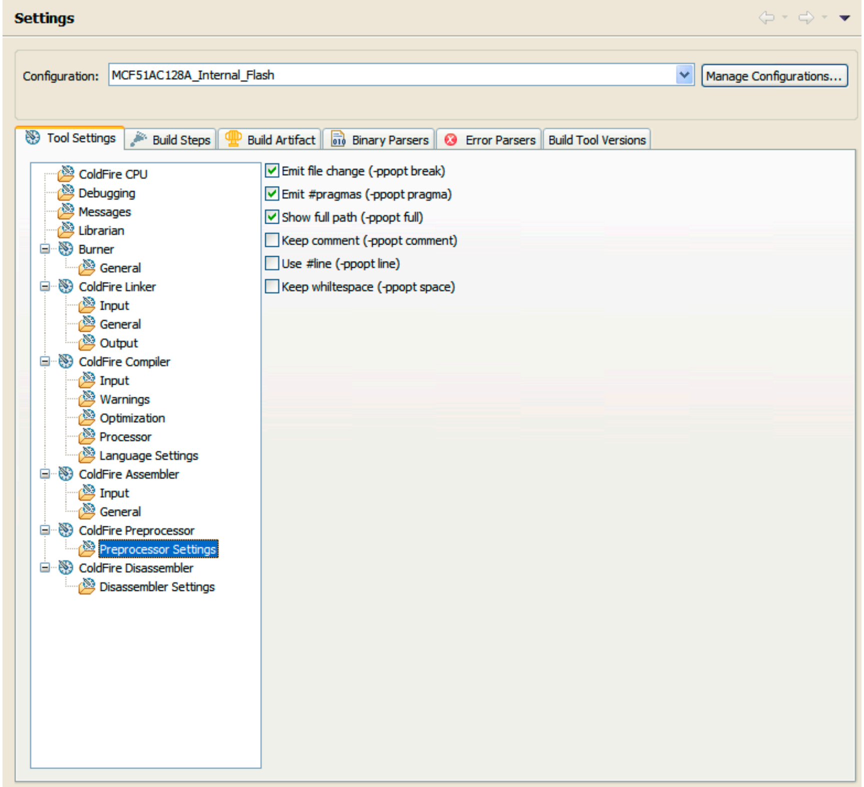
Option	Description
Command	Shows the location of the disassembler executable file
All options	Shows the actual command line the preprocessor will be called with
Expert Settings Command line pattern	Shows the expert settings command line parameters; default `${COMMAND} ${FLAGS} ${INPUTS}`

ColdFire Preprocessor > Preprocessor Settings

Use this panel to specify preprocessor behavior.

Figure 3.130 shows the **Preprocessor** panel.

Figure 3.130 Tool Settings — ColdFire Preprocessor > Preprocessor Settings



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

☑ Emit file change (-ppopt break)
☑ Emit #pragmas (-ppopt pragma)
☑ Show full path (-ppopt full)
☐ Keep comment (-ppopt comment)
☐ Use #line (-ppopt line)
☐ Keep whitespace (-ppopt space)

Table 3.122 lists and describes the preprocessor options for ColdFire.

Table 3.122 Tool Settings — ColdFire Compiler > Preprocessor Options

Option	Description
Emit file change (-ppopt break)	Check to notify file changes (or #line changes) appear in the output.
Emit #pragmas (-ppopt pragma)	Check to show pragma directives in the preprocessor output. Essential for producing reproducible test cases for bug reports.
Show full path (-ppopt full)	Check to display file changes in comments (as before) or in #line directives.

Table 3.122 Tool Settings — ColdFire Compiler > Preprocessor Options (*continued*)

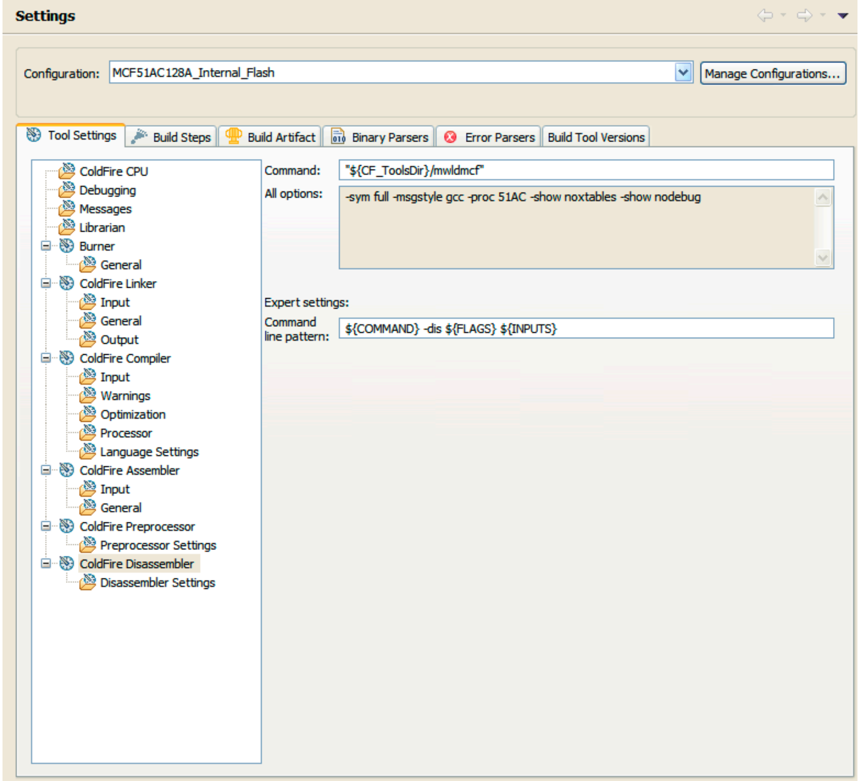
Option	Description
Keep comment (-ppopt comment)	Check to display comments in the preprocessor output.
Use #include line (-ppopt line)	Check to display file changes in comments (as before) or in #line directives.
Keep whitespace (-ppopt nospace)	Check to copy whitespaces in preprocessor output. This is useful for keeping the starting column aligned with the original source, though the compiler attempts to preserve space within the line. This does not apply when macros are expanded.

ColdFire Disassembler

Use this panel to specify the command, options, and expert settings for **ColdFire Disassembler.**

Figure 3.131 shows the **ColdFire Disassembler** page.

Figure 3.131 Tool Settings — ColdFire Disassembler


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Configuration: MCF51AC128A_Internal_Flash

Command: "${CF_ToolsDir}/mwldmcf"

All options: -sym full -msgstyle gcc -proc 51AC -show noxtables -show nodebug

Expert settings:

Command line pattern: ${COMMAND} -dis ${FLAGS} ${INPUTS}

Table 3.123 lists and describes the ColdFire disassembler options.

Table 3.123 Tool Settings — Linker Options

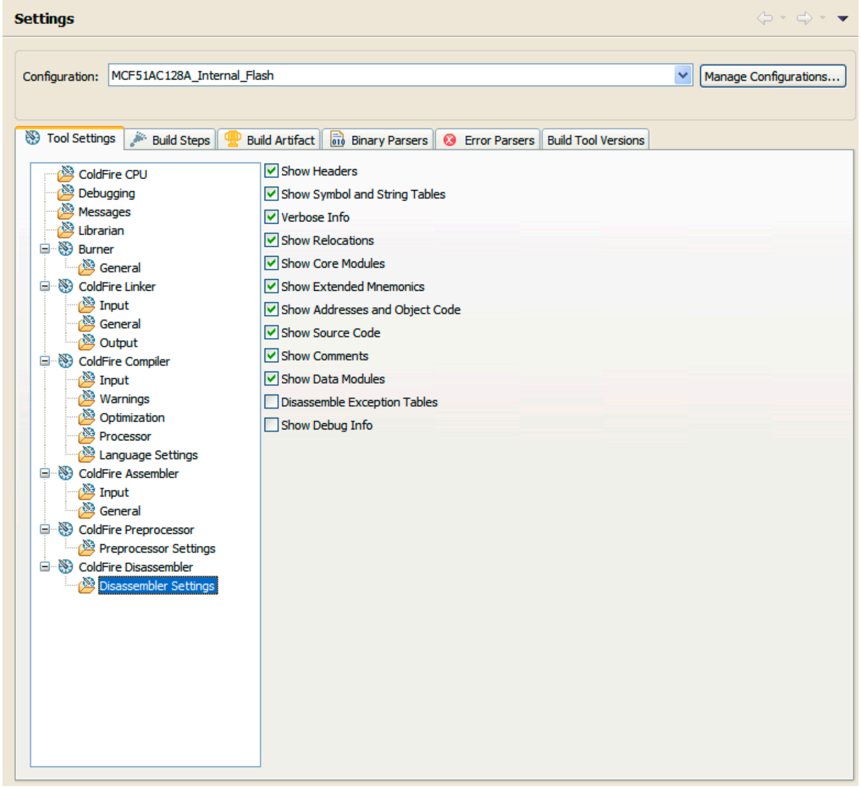
Option	Description
Command	Shows the location of the disassembler executable file
All options	Shows the actual command line the linker will be called with
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is`${COMMAND} -dis ${FLAGS} ${INPUTS}`

ColdFire Disassembler > Disassembler Settings

Use this panel to control how the disassembler formats the listing file, as well as error and warning messages. You can specify verbosity of messages, whether to show headers, core modules, extended mnemonics, addresses, object or source code, ldata modules, exception tables, and debug information.

Figure 3.132 shows the **ColdFire Disassembler** settings.

Figure 3.132 Tool Settings — ColdFire Disassembler Settings



Configuration: MCF51AC128A_Internal_Flash

☑ Show Headers
☑ Show Symbol and String Tables
☑ Verbose Info
☑ Show Relocations
☑ Show Core Modules
☑ Show Extended Mnemonics
☑ Show Addresses and Object Code
☑ Show Source Code
☑ Show Comments
☑ Show Data Modules
☐ Disassemble Exception Tables
☐ Show Debug Info

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ColdFire CPU
Debugging
Messages
Librarian
Burner
General
ColdFire Linker
Input
General
Output
ColdFire Compiler
Input
Warnings
Optimization
Processor
Language Settings
ColdFire Assembler
Input
General
ColdFire Preprocessor
Preprocessor Settings
ColdFire Disassembler
Disassembler Settings

Table 3.124 lists and describes the ColdFire disassembler settings.

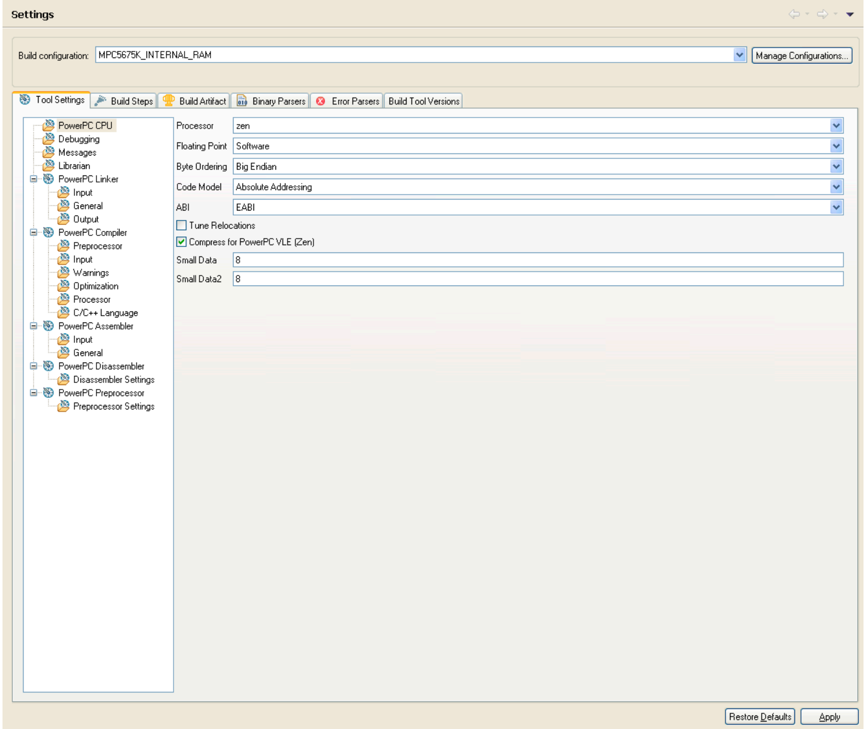
Table 3.124 Tool Settings — ColdFire Disassembler Options

Option	Description
Show Headers	Check to display headers in the listing file; disassembler writes listing headers, titles, and subtitles to the listing file
Show Symbol and String Tables	Check to display symbol and string tables directives to the listing file
Verbose Info	Check to shows each command line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit
Show Core Modules	Check to show core modules in the listing file
Show Extended Mnemonics	Check to show the extended mnemonics in the listing file
Show Addresses and Object Code	Check to show the addresses and object code in the listing file
Show Source Code	Check to show the source code in the listing file
Show Comments	Check to show the comments in the listing file
Show Data Modules	Check to show the data modules in the listing file
Disassemble Exception Tables	Check to disassemble exception tables in the listing file
Show Debug Info	Check to generate symbolic information for debugging the build target

Build Properties for Power Architecture

The **Properties for *<project>*** window shows the corresponding build properties for a Power Architecture project (Figure 3.133).

Figure 3.133 Build Properties - Power Architecture


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Processor: zen
Floating Point: Software
Byte Ordering: Big Endian
Code Model: Absolute Addressing
ABI: EABI
☐ Tune Relocations
☑ Compress for PowerPC VLE (Zen)
Small Data: 8
Small Data2: 8

Table 3.125 lists the build properties specific to developing software for Power Architecture. The properties that you specify in these panels apply to the selected build tool on the **Tool Settings** page of the **Properties for <*project*>** window.

Table 3.125 Build Properties for Power Architecture

Build Tool	Build Properties Panels
PowerPC CPU	PowerPC CPU
Debugging	Debugging
Messages	Messages
Librarian	Librarian

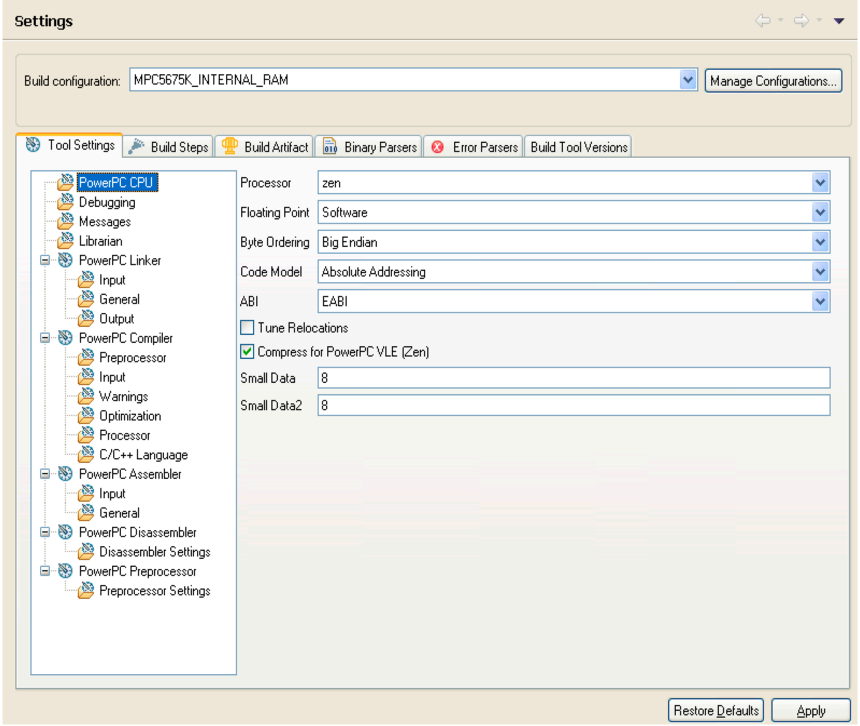
Table 3.125 Build Properties for Power Architecture (*continued*)

Build Tool	Build Properties Panels
PowerPC Linker	PowerPC Linker > Input
	PowerPC Linker > General
	PowerPC Linker > Output
PowerPC Compiler	PowerPC Compiler > Preprocessor
	PowerPC Compiler > Input
	PowerPC Compiler > Warnings
	PowerPC Compiler > Optimization
	PowerPC Compiler > Processor
	PowerPC Compiler > C/C++ Language
PowerPC Assembler	PowerPC Assembler > Input
	PowerPC Assembler > General
PowerPC Disassembler	PowerPC Disassembler > Disassembler Settings
PowerPC Disassembler	PowerPC Preprocessor > Preprocessor Settings

PowerPC CPU

Use this panel to specify the CPU type, and the memory model that the architecture uses. The build tools (compiler, linker, and assembler) then use the properties set in this panel to generate CPU-specific code.

Figure 3.134 Tool Settings — PowerPC CPU



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Processor: zen
Floating Point: Software
Byte Ordering: Big Endian
Code Model: Absolute Addressing
ABI: EABI
☐ Tune Relocations
☑ Compress for PowerPC VLE (Zen)
Small Data: 8
Small Data2: 8

Table 3.126 lists and describes the PowerPC CPU options.

Table 3.126 Tool Settings — PowerPC CPU

Option	Description
Processor	Lists the processor families supported by the Power Architecture compiler. When you select a processor from this list, the compiler generates code that makes use of any of its hardware features or special instructions. For more detailed information on the features of each processor, refer to its reference manual document.
Floating Point	Define how the compiler handles floating-point operations it encounters in the source code. • **Software** — Select to have the compiler emulate floating-point operations by calling functions that perform floating-point math. The C runtime library contains the functions the compiler invokes. If you use software floating-point emulation, you must include the appropriate C runtime library in your project. Enabling this option without including the appropriate C runtime library causes link errors. • **Hardware** — Select to have the compiler handle floating-point operations by generating instructions for the hardware floating-point unit. Do not select this option if your target processor does not have a hardware floating-point unit. • **None** — Select to disable floating-point support. • **SPFP** — Select to have the compiler handle single-precision floating-point operations by generating instructions for the e500-EFPU floating point unit, and perform double-precision floating-point operations by calling functions that perform double-precision floating-point math. Do not select this option if your target processor does not have a e500-EFPU floating-point unit. • **DPFP** — Select to have the compiler handle both single- and double-precision floating-point operations by generating instructions for the e500 DPFP APU (Double-Precision Floating-Point Auxiliary Processing Unit). Do not select this option if your target processor does not have a DPFPunit. **Default**: Software If the selected processor does not handle a floating-point exception, you should select `None` or `Software` floating-point support.

Table 3.126 Tool Settings — PowerPC CPU (*continued*)

Option	Description
Byte Ordering	Enables you to select big-endian or little-endian byte ordering. • **Big Endian** — Select to generate object code and links an executable image that uses big-endian byte ordering. This is the default setting for the compiler and linker. If you choose big-endian byte ordering, within a given multi-byte numeric representation, the most significant byte has the lowest address; the word is stored big-end-first. • **Small Endian** — Select to generates object code and links an executable image that uses little-endian byte ordering. If you choose little endian byte ordering, within a given multi-byte numeric representation, bytes at lower addresses have lower significance; the word is stored little-end-first.
Code Model	Enables you to select the addressing mode for the binary generated by the current build target. • **Absolute Addressing** — Select to instruct the build tools to generate a non-relocatable binary. • **SDA Based PIC/PID Addressing** — Select to instruct the build tools to generate a relocatable binary that uses position independent-code (PIC)/position-independent-data (PID) addressing. The resulting binary can be loaded at any address.
ABI	Enables you to select the Application Binary Interface (ABI) the compiler and assembler use for function calls and structure layout. • **EABI** — Converts a 14-bit branch relocation to a 24-bit branch relocation only if the 14-bit relocation cannot reach the calling site from the original relocation. • **System V ABI** — Changes the absolute addressed references of data from code to use a small data register (such as r13) instead of r0; absolute code is changed to code references to use the PC relative relocations. • **SuSE** — Use the SuSE® Linux ABI with GNU extensions. • **YellowDog** — Use the YellowDog™ Linux ABI with GNU extensions. • **SDA PIC/PID** — Use position-independent addressing executable code and data.
Tune Relocations	Pertains to object relocation and is available for just the above mentioned application binary interfaces.

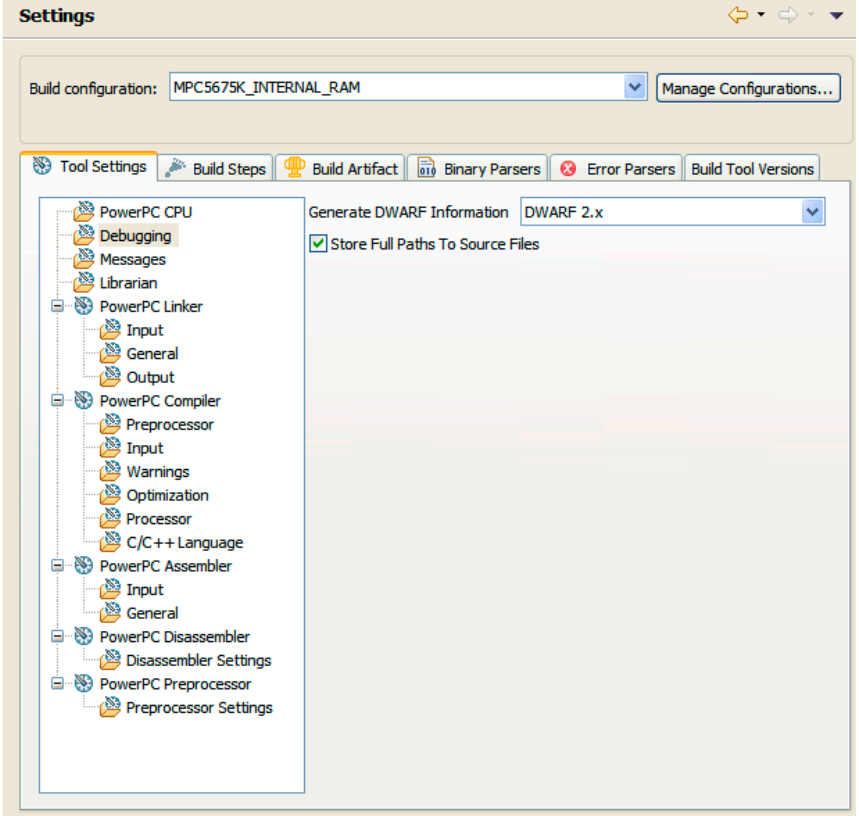
Table 3.126 Tool Settings — PowerPC CPU (*continued*)

Option	Description
Compress for PowerPC VLE (Zen)	
Small Data	Specify the threshold size (in bytes) for an item to be considered small data by the linker. The linker stores small data items in the Small Data address space. Data in the Small Data address space can be accessed more quickly than data in the normal address space.
Small Data2	Specify the threshold size (in bytes) for an item to be considered small data by the linker. The linker stores read-only small data items in the Small Data2 address space. Constant data in the Small Data2 address space can be accessed more quickly than data in the normal address space.

Debugging

Use this panel to specify the whether to generate symbolic information for debugging the build target (Figure 3.135).

Figure 3.135 Tool Settings — Debugging



Settings
Build configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings

Build configuration: MPC5675K_INTERNAL_RAM

Generate DWARF Information: DWARF 2.x

☑ Store Full Paths To Source Files

Table 3.127 lists and describes the debugging options.

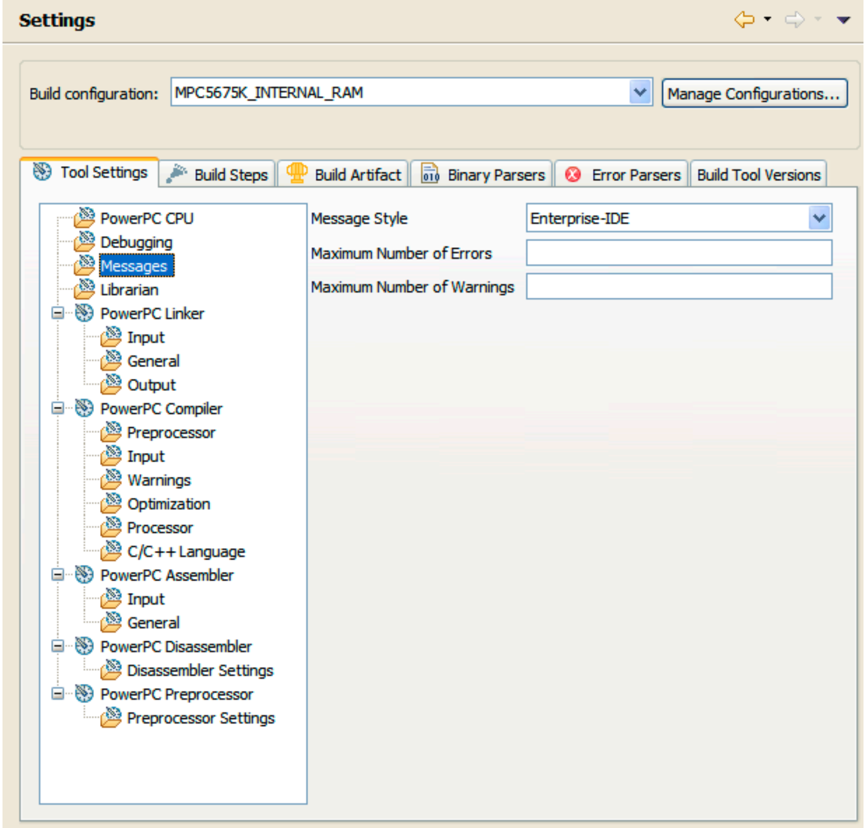
Table 3.127 Tool Settings — Debugging

Option	Description
Generate DWARF Information	Select the version of the Debug With Arbitrary Record Format (DWARF) debugging information format the compiler and assembler generates. If in doubt about the DWARF version to use, you can use the default setting of DWARF 2.x. The linker ignores debugging information that is not in the selected format.
Store Full Paths to Source Files	Store absolute paths of source files instead of relative paths.

Messages

Use this panel to specify the whether to generate symbolic information for debugging the build target (Figure 3.136).

Figure 3.136 Tool Settings — Messages


Settings
Build configuration:
MPC5675K_INTERNAL_RAM
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings

Message Style: Enterprise-IDE

Maximum Number of Errors:

Maximum Number of Warnings:

Table 3.128 lists and describes the message options.

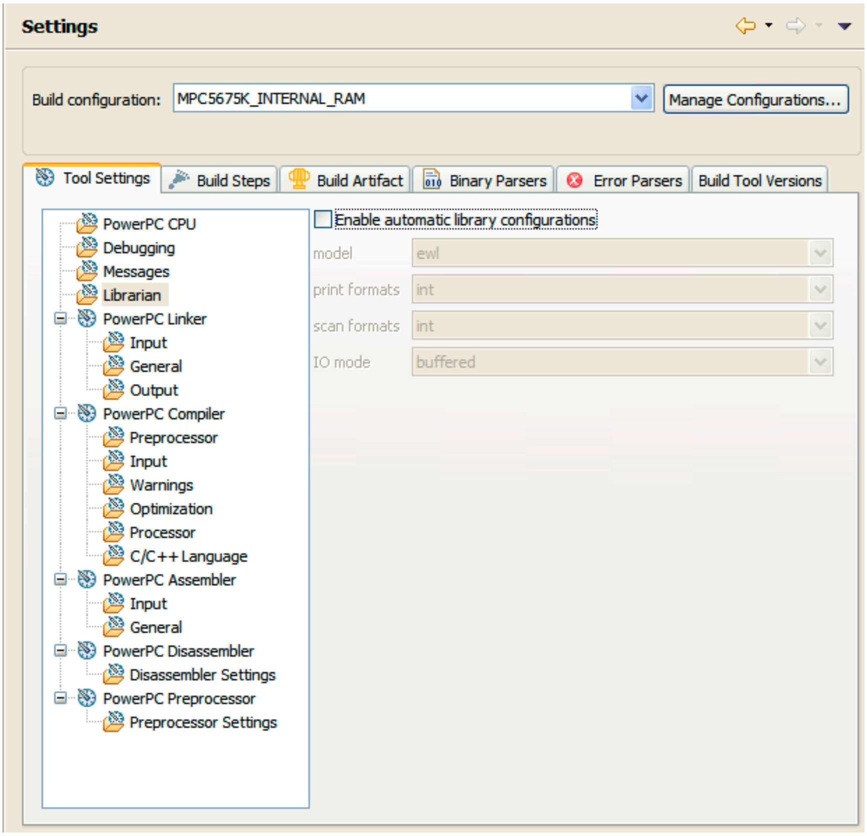
Table 3.128 Tool Settings — Messages Options

Option	Description
Message Style	List options to select message style. • **GCC(default)** — Uses the message style of the Gnu Compiler Collection tools • **MPW** — Uses the Macintosh Programmer's Workshop (MPW®) message style • **Standard** — Uses the standard message style • **IDE** — Uses context-free machine parseable message style • **Enterprise-IDE —** Uses CodeWarrior's Integrated Development Environment (IDE) message style. • **Parseable** — Uses parseable message style.
Maximum Number of Errors	Specify the number of errors allowed until the application stops processing.
Maximum Number of Warnings	Specify the maximum number of warnings.

Librarian

Use this panel to select whether the linker will identify standard libraries (Figure 3.137).

Figure 3.137 Tool Settings — Librarian


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings

Build configuration: MPC5675K_INTERNAL_RAM

☐ Enable automatic library configurations

model: ewl

print formats: int

scan formats: int

IO mode: buffered

Table 3.129 lists and describes the librarian options.

Table 3.129 Tool Settings — Librarian Options

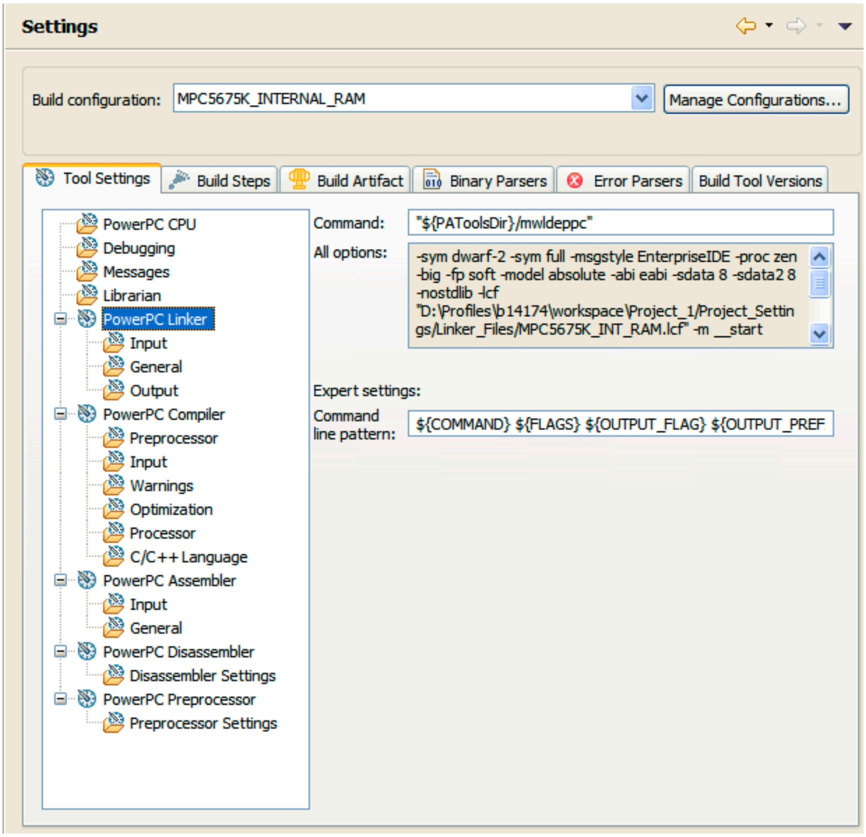
Option	Description
Enable automatic library configurations	Select to let the compiler identify standard libraries.
Model	Select a standard complying or EWL model from the drop-down list. EWL lets you precisely define the I/O operations. EWL drastically reduces the size of executables as you explicitly select the appropriate I/O behavior. Options are: `ewl, c9x, ewl_c++,` and `c9x_c++.`
Print formats	Select the print formats from the drop-down list. The available options are: `int, int_FP, int_LL,` and `int_LL_FP`.
Scan formats	Select the scan formats from the drop-down list. The available options are: `int, int_FP, int_LL,` and `int_LL_FP`.
IO Mode	Select the input-output mode from the drop-down list. The available options are: `raw` and `buffered`.

PowerPC Linker

Use this panel to specify PowerPC linker behavior. You can specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the input, general, and output settings.

Figure 3.138 shows the **PowerPC Linker** settings.

Figure 3.138 Tool Settings — PowerPC Linker


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Expert settings:

Build configuration: MPC5675K_INTERNAL_RAM

Command: "${PAToolsDir}/mwldeppc"

All options: -sym dwarf-2 -sym full -msgstyle EnterpriseIDE -proc zen -big -fp soft -model absolute -abi eabi -sdata 8 -sdata2 8 -nostdlib -lcf "D:\Profiles\b14174\workspace\Project_1/Project_Settings/Linker_Files/MPC5675K_INT_RAM.lcf" -m __start

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREF

Table 3.130 lists and describes the linker options for PowerPC.

Table 3.130 Tool Settings — PowerPC Linker

Option	Description
Command	Shows the location of the linker executable file. Default: `${PAToolsDir}/mwldeppc`
All options	Shows the actual command line the linker will be called with.

Table 3.130 Tool Settings — PowerPC Linker (*continued*)

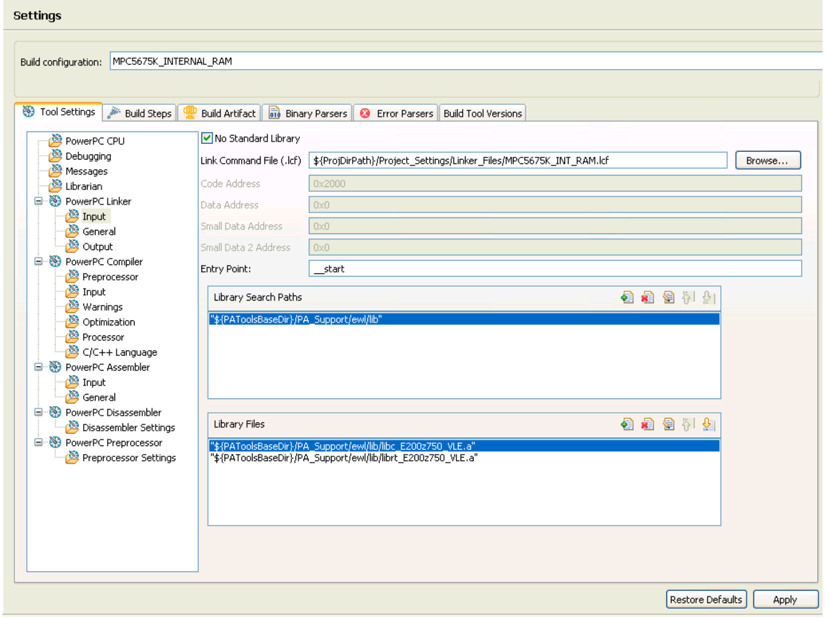
Option	Description
Expert settings	Shows the expert settings command line parameters.
Command line pattern	Default: `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

PowerPC Linker > Input

Use this panel to specify files the **PowerPC Linker** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.139 shows the **Input** panel.

Figure 3.139 Tool Settings — PowerPC Linker > Input


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Browse...
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

☑ No Standard Library

Link Command File (.lcf) ${ProjDirPath}/Project_Settings/Linker_Files/MPC5675K_INT_RAM.lcf

Code Address 0x2000

Data Address 0x0

Small Data Address 0x0

Small Data 2 Address 0x0

Entry Point: __start

Library Search Paths

"${PAToolsBaseDir}/PA_Support/ewl/lib"

Library Files

"${PAToolsBaseDir}/PA_Support/ewl/lib/libc_E200z750_VLE.a"
"${PAToolsBaseDir}/PA_Support/ewl/lib/librt_E200z750_VLE.a"

Table 3.131 lists and describes the input options for PowerPC.

Table 3.131 Tool Settings — PowerPC Linker > Input

Option	Description
No Standard Library (-nostdlib)	Select if there is no standard library attached
Link Command File (.lcf)	Consists of three kinds of segments, which must be in this order: • A memory segment, which begins with the MEMORY{} directive • Optional closure segments, which begin with the FORCE_ACTIVE{}, KEEP_SECTION{}, or REF_INCLUDE{} directives • A sections segment, which begins with the SECTIONS{} directive
Code Address	Specifies the location in memory where the executable code resides. The possible addresses depend on your target hardware platform and how the memory is mapped.
Data Address	Enables you to specify the address for global data.
Small Data Address	Enables you to specify the RAM address for the first small data section. This address must not conflict with the target-hardware memory map; target hardware must support this address.
Small Data2 Address	Enables you to specify the RAM address for the second small data section. This address must not conflict with the target-hardware memory map; target hardware must support this address.
Entry Point	Enables you to specify the program starting point - the function that the linker uses first when you launch the program. This default function (in file __start.c) is bootstrap/glue code that sets up the EABI environment, then calls function main().

Table 3.131 Tool Settings — PowerPC Linker > Input

Option	Description
Library Search Paths (-L +path)	Enables you to specify the search pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.
Library Files ?(-l +file)	Enables you to specify the pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.

Table 3.132 lists and describes the toolbar buttons that help work with the libraries and the additional object file search paths.

Table 3.132 Search Paths Toolbar Buttons

Button	Description
➕	**Add —** Click to open the **Add directory path** dialog box (Figure 3.140) and specify the object file search path.
✖	**Delete —** Click to delete the selected object file search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.141) and update the selected object file search path.
⇧	**Move up —** Click to move the selected object file search path one position higher in the list.
⇩	**Move down —** Click to move the selected object file search path one position lower in the list.

Figure 3.140 shows the **Add directory path** dialog box.

Figure 3.140 Add directory path Dialog Box

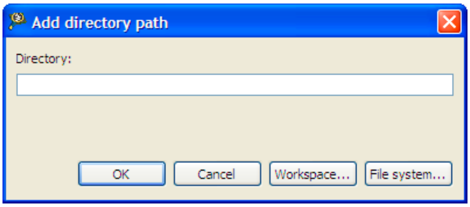
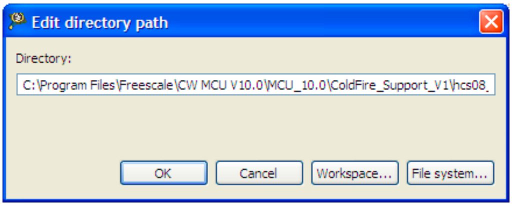


Figure 3.141 shows the **Edit directory path** dialog box.

Figure 3.141 Edit directory path Dialog Box



C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

- **OK —** Click to confirm the action and exit the dialog box.
- **Cancel —** Click to cancel the action and exit the dialog box.
- **Workspace —** Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system —** Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

Table 3.133 lists and describes the toolbar buttons that help work with the libraries and the additional object files.

Table 3.133 Libraries Toolbar Buttons

Button	Description
➕	**Add —** Click to open the **Add file path** dialog box (Figure 3.142) and specify location of the library you want to add.
✖	**Delete —** Click to delete the selected library path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.

Table 3.133 Libraries Toolbar Buttons (*continued*)

Button	Description
✎	**Edit —** Click to open the **Edit file path** dialog box (Figure 3.143) and update the selected path.
↑	**Move up —** Click to move the selected path one position higher in the list.
↓	**Move down —** Click to move the selected path one position lower in the list.

Figure 3.142 shows the **Add file path** dialog box.

Figure 3.142 Tool Settings — Linker > Libraries - Add file path Dialog Box

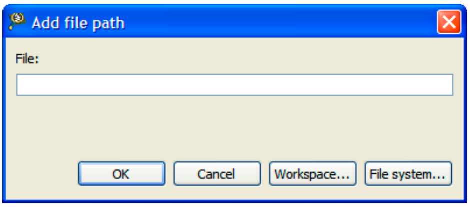
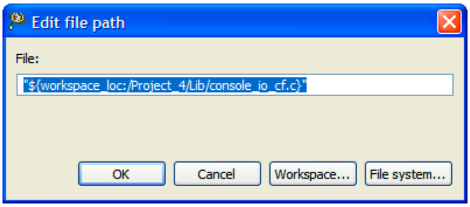


Figure 3.143 shows the **Edit file path** dialog box.

Figure 3.143 Tool Settings — Linker > Libraries - Edit file path Dialog Box



File: "${workspace_loc:/Project_4/Lib/console_io_cf.c}"

The buttons in the **Add file path and Edit file path** dialog boxes help work with the file paths.

- **OK —** Click to confirm the action and exit the dialog box.
- **Cancel —** Click to cancel the action and exit the dialog box.
- **Workspace —** Click to display the **File Selection** dialog box and specify the file path. The resulting path, relative to the workspace, appears in the appropriate list.

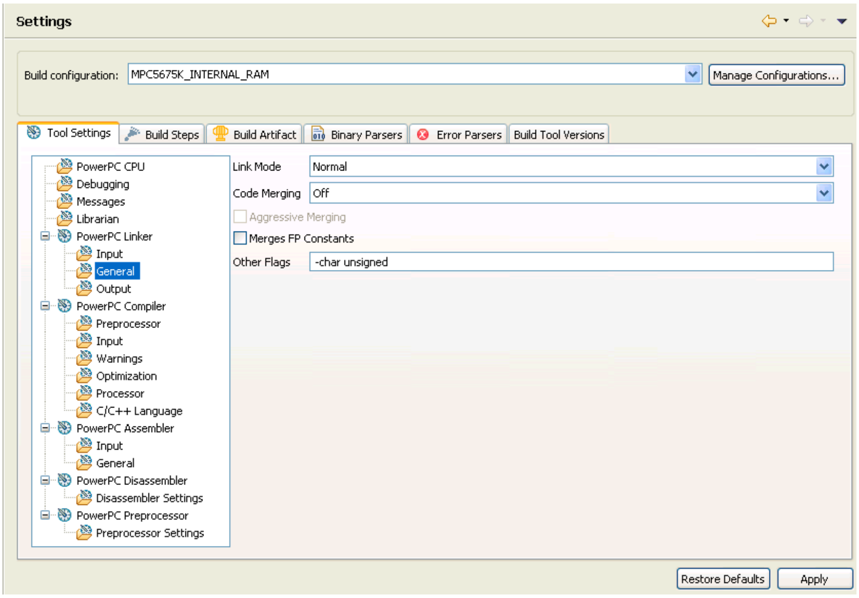
- **File system —** Click to display the **Open** dialog box and specify the file path. The resulting absolute path appears in the appropriate list.

PowerPC Linker > General

Use this panel to specify the memory model that the architecture uses. The build tools (compiler, linker, and assembler) use the properties that you specify.

Figure 3.144 shows the **General** settings.

Figure 3.144 Tool Settings — General



Build configuration: MPC5675K_INTERNAL_RAM

Link Mode: Normal

Code Merging: Off

☐ Aggressive Merging

☐ Merges FP Constants

Other Flags: -char unsigned

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.134 lists and describes the memory model options for PowerPC.

Table 3.134 Tool Settings — PowerPC Linker > General

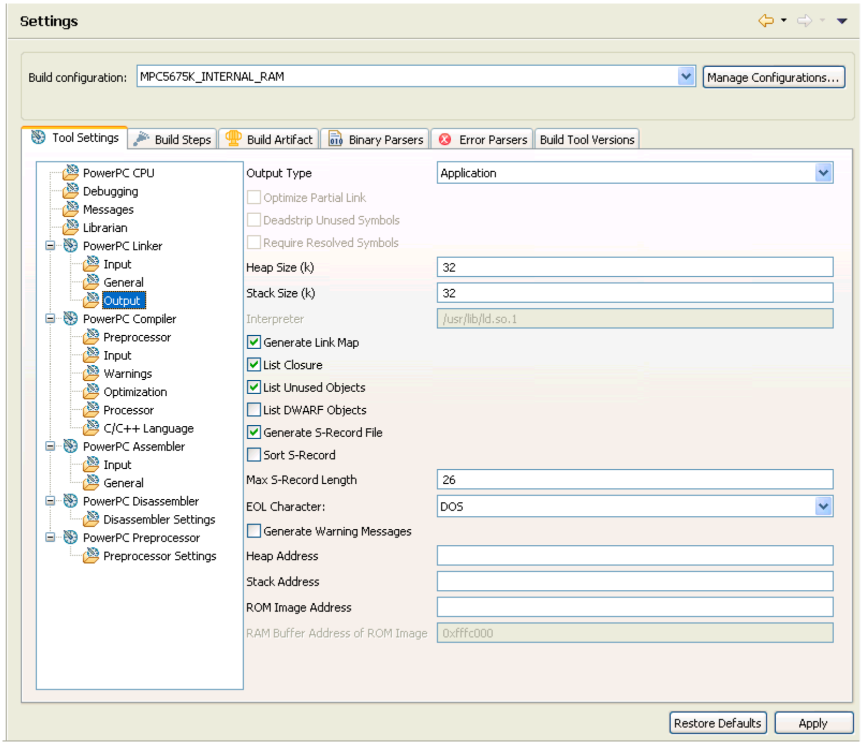
Option	Description
Link Mode	Specifies how much memory the linker uses to write output to the hard disk. • **Normal** — Writes to a 512-byte buffer, then writes the buffer to disk. • **Use Less RAM** — Writes output file directly to disk, without using a buffer. • **Use More RAM** — Writes each segment to its own buffer, then flushes all buffers to the disk. Linking requires enough RAM space for all input files and numerous housekeeping structures. Normal is the best choice for most projects; **Use More RAM** is appropriate for small projects.
Code Merging	Controls merging optimization. • **Off** • **Safe Functions** • **All Functions** Checking **Off** deactivates the **Aggressive Merging** checkbox.
Aggressive Merging	Check to implement aggressive merging. This checkbox is active only if **Safe Functions** or **All Functions** is selected in the **Code Merging** drop-down list. Clear if you do not want to implement aggressive merging.
Merges FP Constants	Check to let the linker automatically merge floating-point constants. Clear if you do not want to enable floating-point constants for automatic merging.
Other Flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

PowerPC Linker > Output

Use this panel to specify the output settings for the PowerPC linker.

Figure 3.145 shows the **Output** panel.

Figure 3.145 Tool Settings — PowerPC Linker > Output


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Output Type: Application
☐ Optimize Partial Link
☐ Deadstrip Unused Symbols
☐ Require Resolved Symbols
Heap Size (k): 32
Stack Size (k): 32
Interpreter: /usr/lib/ld.so.1
☑ Generate Link Map
☑ List Closure
☑ List Unused Objects
☐ List DWARF Objects
☑ Generate S-Record File
☐ Sort S-Record
Max S-Record Length: 26
EOL Character: DOS
☐ Generate Warning Messages
Heap Address:
Stack Address:
ROM Image Address:
RAM Buffer Address of ROM Image: 0xfffc000

Table 3.135 lists and describes the output settings for PowerPC linker.

Table 3.135 Tool Settings — PowerPC Linker > Output

Option	Description
Output Type	User choose to build an application, a library, or a partial link project.

Table 3.135 Tool Settings — PowerPC Linker > Output (*continued*)

Option	Description
Optimize Partial Link	Check to specify direct downloading of the partial link output. Clear if you want the output file to remain as if you passed the -r argument in the command line. This checkbox appears only if the **Output Type** drop-down list specifies **Partial Link**.
Deadstrip Unused Symbols	Check to let the linker deadstrip all unused symbols. This reduces program size, by removing symbols that neither the main entry point or force-active entry points reference. Clear if you do not want the linker to deadstrip unused symbols. This checkbox appears only if the **Optimize Partial Link** is checked.
Require Resolved Symbols	Check if the linker must resolve all symbols in the partial link and if your RTOS does not allow unresolved symbols. Clear if the linker does not have to resolve all symbols of the partial link. This checkbox appears only if the **Optimize Partial Link** is checked.
Heap Size (k)	Specify kilobytes of RAM allocated for the heap, which your program uses if it calls malloc or new. Combined heap/ stack allocation must not exceed available RAM. This checkbox appears only if the **Output Type** drop-down list specifies **Application**.
Stack Size (k)	Specify kilobytes of RAM allocated for the stack. Combined heap/stack allocation must not exceed available RAM. This checkbox appears only if the **Output Type** drop-down list specifies **Application**.
Interpreter	The linker to use an interpreter file.
Generate Link Map	Check to let the linker generate a link map - showing every object/function definition and address, memory map of sections, and values of linker-generated symbols. Activates subordinate checkboxes. Clear if you do not want the linker to generate a map file. If you used a non-CodeWarrior compiler to build the relocatable file, the map file also lists unused but unstripped symbols. Map files have the extension .MAP

Table 3.135 Tool Settings — PowerPC Linker > Output (*continued*)

Option	Description
List Closure	Check if you want the map file list all functions that the program starting point calls. Clear if you do not want the map file list functions that the program starting point calls. This checkbox is active only if the **Generate Link Map** checkbox is checked.
List Unused Objects	Check if you want the map file to list unused objects; useful for revealing that objects you expected to be used are not. Clear if you do not want the map to list unused objects. This checkbox is active only if the **Generate Link Map** checkbox is checked.
List DWARF Objects	Check if you want the list map lists all DWARF debugging objects in section area. Clear if you do not want the map file to list DWARF debugging objects. This checkbox is active only if the **Generate Link Map** checkbox is checked.
Generate S-Record File	Check to generate an S3 S-record file, based on the application object image and activates the subordinate elements. Clear if you do not want to generate an S-record file. The name extension of the S-record file is .mot
Sort S-Record	Check to sort generated S-record files in ascending address order. Clear if you do not want to sort the S-record files. This checkbox is active only if the **Generate S-Record File** checkbox is checked.
Max S-Record Length	Specifies maximum S-record length (256 bytes or fewer) for the system. (For a non-CodeWarrior tool, you may need to reduce this value.) This checkbox is active only if the **Generate S-Record File** checkbox is checked.

Table 3.135 Tool Settings — PowerPC Linker > Output (*continued*)

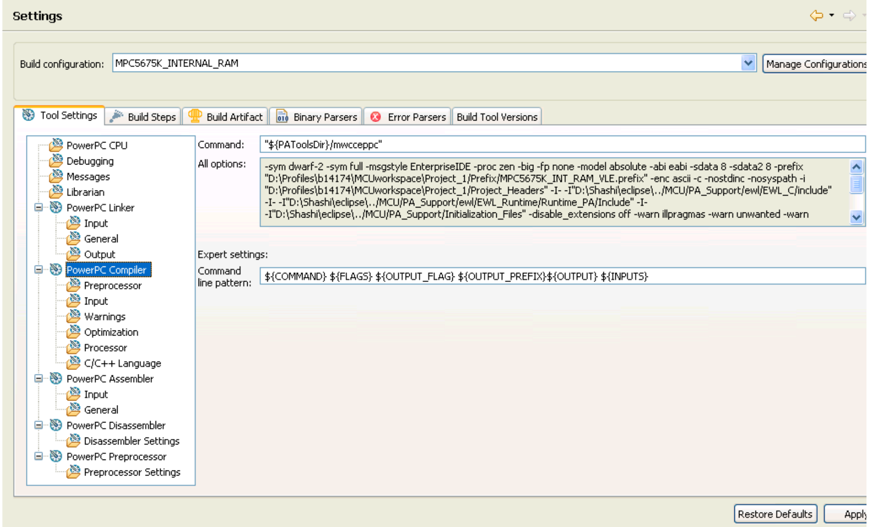
Option	Description
EOL Character	Specifies the end-of-line character for the S-record file:. • Mac - <cr> • DOS - <cr> <lf> • Unix - <lf> This drop-down list is active only if the **Generate S-Record File** checkbox is checked.
Generate Warning Messages	Check if you want the linker to generate warning messages.
Heap Address	Specifies the memory location for program heap where you can enter the RAM address of the bottom of the heap.
Stack Address	Specifies memory location for program stack where you can enter the RAM address for the top of the stack.
ROM Image Address	Specifies the flash ROM destination address for your binary.
RAM Buffer Address of ROM Image	Specifies the RAM buffer address for the ROM image. This option is active only if a value is specified in the **ROM Image Address** textbox. For the CodeWarrior flash programmer, the ROM image address and the RAM buffer address must be the same.

PowerPC Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the PowerPC Compiler tree control includes the general and the file search path settings.

Figure 3.146 shows the PowerPC Compiler settings.

Figure 3.146 Tool Settings — PowerPC Compiler


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Manage Configurations
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Command: "${PAToolsDir}/mwcceppc"

All options:

```
-sym dwarf-2 -sym full -msgstyle EnterpriseIDE -proc zen -big -fp none -model absolute -abi eabi -sdata 8 -sdata2 8 -prefix
"D:\Profiles\b14174\MCUworkspace\Project_1/Prefix/MPC5675K_INT_RAM_VLE.prefix" -enc ascii -c -nostdinc -nosyspath -i
"D:\Profiles\b14174\MCUworkspace\Project_1/Project_Headers" -I- -I"D:\Shashi\eclipse\../MCU/PA_Support/ewl/EWL_C/include"
-I- -I"D:\Shashi\eclipse\../MCU/PA_Support/ewl/EWL_Runtime/Runtime_PA/Include" -I-
-I"D:\Shashi\eclipse\../MCU/PA_Support/Initialization_Files" -disable_extensions off -warn illpragmas -warn unwanted -warn
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.136 lists and describes the compiler options for PowerPC.

Table 3.136 Tool Settings — PowerPC Compiler

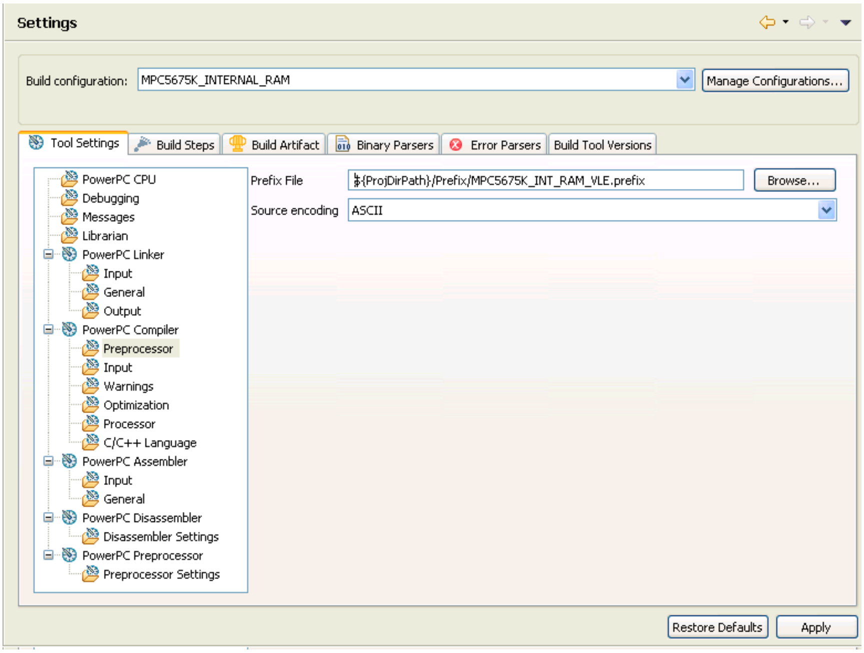
Option	Description
Command	Shows the location of the linker executable file. Default value is “`${PAToolsDir}/mwcceppc`”
All options	Shows the actual command line the compiler will be called with.
Expert settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

PowerPC Compiler > Preprocessor

Use this panel to specify preprocessor behavior.

Figure 3.147 shows the **Preprocessor** panel.

Figure 3.147 Tool Settings — PowerPC Compiler > Preprocessor Settings



Build configuration: MPC5675K_INTERNAL_RAM

Prefix File: ${ProjDirPath}/Prefix/MPC5675K_INT_RAM_VLE.prefix

Source encoding: ASCII

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Browse...
Restore Defaults
Apply

Table 3.137 lists and describes the preprocessor options for PowerPC.

Table 3.137 Tool Settings — PowerPC Compiler > Preprocessor

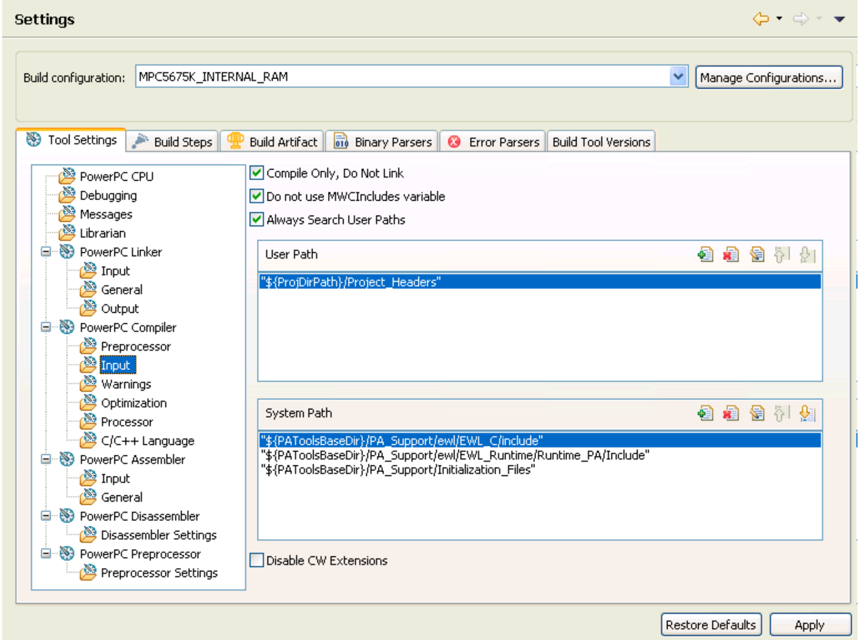
Option	Description
Prefix File	Specifies a file automatically included in all project assembly files.
Source encoding	Lets you specify the default encoding of source files. The compiler recognizes Multibyte and Unicode source text. To replicate the obsolete option Multi-Byte Aware, set this option to System or Autodetect. Additionally, options that affect the preprocess request appear in this panel.

PowerPC Compiler > Input

Use this panel to specify additional files the PowerPC Compiler should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.148 shows the **Input** panel.

Figure 3.148 Tool Settings — PowerPC Compiler > Input


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
User Path
System Path
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

☑ Compile Only, Do Not Link
☑ Do not use MWCIncludes variable
☑ Always Search User Paths

User Path
"${ProjDirPath}/Project_Headers"

System Path
"${PAToolsBaseDir}/PA_Support/ewl/EWL_C/include"
"${PAToolsBaseDir}/PA_Support/ewl/EWL_Runtime/Runtime_PA/Include"
"${PAToolsBaseDir}/PA_Support/Initialization_Files"

☐ Disable CW Extensions

Table 3.138 lists and describes the input options for PowerPC compiler.

Table 3.138 Tool Settings — PowerPC Compiler > Input Options

Option	Description
Compile only, Do Not Link (-c)	Check if you want to compile only and do not want to link the file.
Do not use MWCIncludes variable (-nostdinc)	Check if you do not want to use MWCIncludes variable.
Always Search User Paths (-nosyspath)	Check if you want to always search user paths.

Table 3.138 Tool Settings — PowerPC Compiler > Input Options (*continued*)

Option	Description
User Path (-i)	Lists the available user paths.
System Path	Lists the available system paths.
Disable CW Extensions	Disable the CW features that may be incompatible if user is exporting code libraries form CW to other compiler and/or linkers.

Table 3.139 lists and describes the toolbar buttons that help work with the user and system search paths.

Table 3.139 Search Paths Toolbar Buttons

Button	Description
➕	**Add** — Click to open the **Add directory path** dialog box (Figure 3.149) and specify the search path.
✖	**Delete** — Click to delete the selected search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit** — Click to open the **Edit directory path** dialog box (Figure 3.150) and update the selected search path.
⇧	**Move up** — Click to move the selected search path one position higher in the list
⇩	**Move down** — Click to move the selected search path one position lower in the list

Figure 3.118 shows the **Add directory path** dialog box.

Figure 3.149 Add directory path Dialog Box

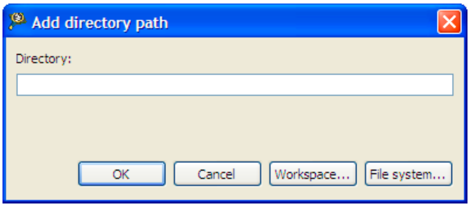
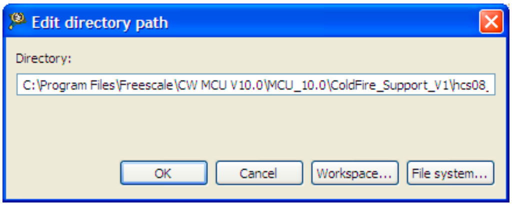


Figure 3.150 shows the **Edit directory path** dialog box.

Figure 3.150 Edit directory path Dialog Box



Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

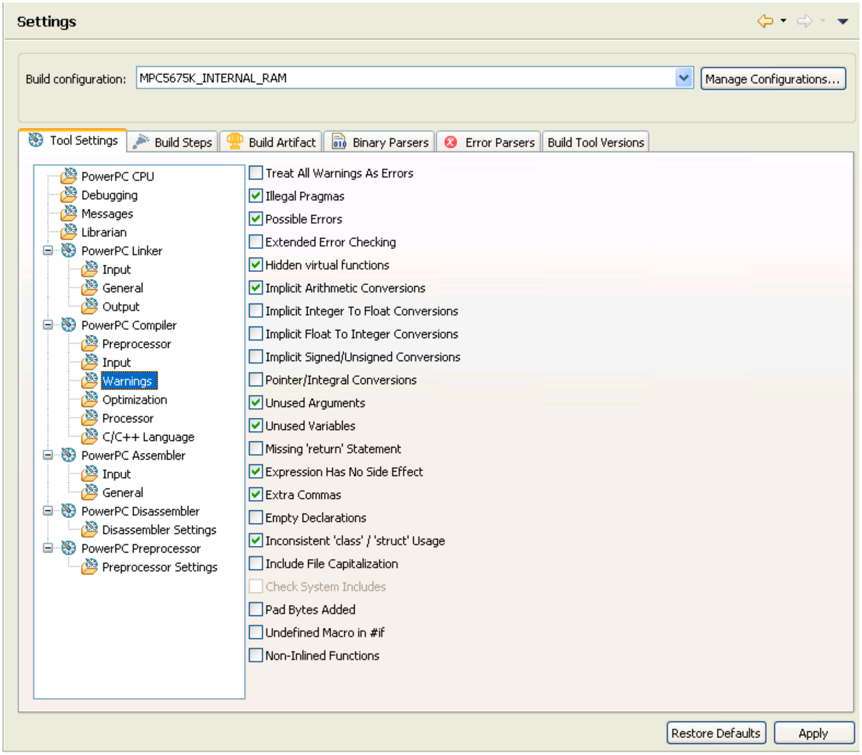
- **OK —** Click to confirm the action and exit the dialog box.
- **Cancel —** Click to cancel the action and exit the dialog box.
- **Workspace —** Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system —** Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

PowerPC Compiler > Warnings

Use this panel to control how the PowerPC compiler formats the listing file, as well as error and warning messages.

Figure 3.151 shows the **Warnings** panel.

Figure 3.151 Tool Settings — PowerPC Compiler > Warnings


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

☐ Treat All Warnings As Errors
☑ Illegal Pragmas
☑ Possible Errors
☐ Extended Error Checking
☑ Hidden virtual functions
☑ Implicit Arithmetic Conversions
☐ Implicit Integer To Float Conversions
☐ Implicit Float To Integer Conversions
☐ Implicit Signed/Unsigned Conversions
☐ Pointer/Integral Conversions
☑ Unused Arguments
☑ Unused Variables
☐ Missing 'return' Statement
☑ Expression Has No Side Effect
☑ Extra Commas
☐ Empty Declarations
☑ Inconsistent 'class' / 'struct' Usage
☐ Include File Capitalization
☐ Check System Includes
☐ Pad Bytes Added
☐ Undefined Macro in #if
☐ Non-Inlined Functions

Table 3.140 lists and describes the warnings options for PowerPC compiler.

Table 3.140 Tool Settings — PowerPC Compiler > Warnings

Option	Description
Treat All Warnings As Errors	Check to treat all warnings as errors. The compiler will stop if it generates a warning message.
Illegal Pragmas	Check to notify the presence of illegal pragmas.
Possible Errors	Check to suggest possible errors.
Extended Error Checking	Check if you want to do an extended error checking.

Table 3.140 Tool Settings — PowerPC Compiler > Warnings

Option	Description
Hidden virtual functions	Check to generate a warning message if you declare a non-virtual member function that prevents a virtual function, that was defined in a superclass, from being called and is equivalent to pragma `warn_hidevirtual` and the command-line option `-warnings hidevirtual`.
Implicit Arithmentic Conversions	Check to warn of implict arithmetic conversions.
Implicit Integer to Float Conversions	Check to warn of implict conversion of an integer variable to floating-point type.
Implicit Float to Integer Conversions	Check to warn of implict conversions of a floating-point variable to integer type.
Implicit Signed/Unsigned Conversion	Check to enable warning of implict conversions between signed and unsigned variables.
Pointer/Integral Conversions	Check to enable warnings of conversions between pointer and integers.
Unused Arguments	Check to warn of unused arguments in a function.
Unused Variables	Check to warn of unused variables in the code.
Missing ‘return’ Statement	Check to warn of when a function lacks a return statement.
Expression Has No Side Effect	Check to issue a warning message if a source statement does not change the program's state. This is equivalent to the pragma `warn_no_side_effect`, and the command-line option `-warnings unusedexpr`.

Table 3.140 Tool Settings — PowerPC Compiler > Warnings

Option	Description
Extra Commas	Check to issue a warning message if a list in an enumeration terminates with a comma. The compiler ignores terminating commas in enumerations when compiling source code that conforms to the ISO/IEC 9899-1999 ("C99") standard and is equivalent to pragma `warn_extracomma` and the command-line `option -warnings extracomma.`
Empty Declarations	Check to warn of empty declarations.
Inconsistent ‘class’ / ‘struct’ Usage	Check to warn of inconsistent usage of class or struct.
Include File Capitalization	Check to issue a warning message if the name of the file specified in a #include "file" directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps` and the command-line option `-warnings filecaps.`
Check System Includes	Check to issue a warning message if the name of the file specified in a #include <file> directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps_system` and the command-line option `-warnings sysfilecaps.`
Pad Bytes Added	Check to issue a warning message when the compiler adjusts the alignment of components in a data structure and is equivalent to pragma `warn_padding` and the command-line option `-warnings padding.`

Table 3.140 Tool Settings — PowerPC Compiler > Warnings

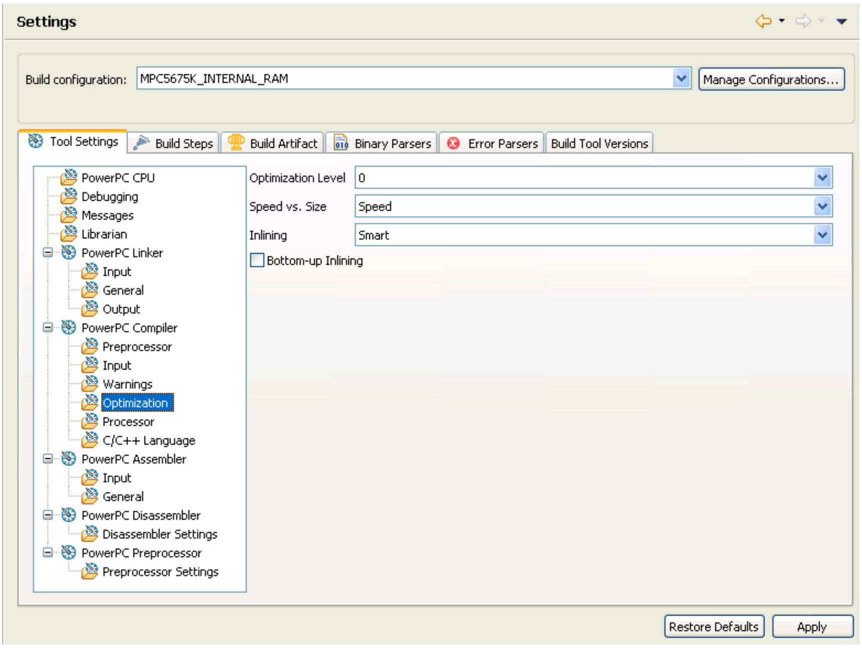
Option	Description
Undefined Macro in #if	Check to issues a warning message if an undefined macro appears in #if and #elif directives and is equivalent to pragma `warn_undefmacro` and the command-line option `-warnings undefmacro.`
Non-Inlined Functions	Check to issue a warning message if a call to a function defined with the inline, __inline__, or __inline keywords could not be replaced with the function body and is equivalent to pragma `warn_notinlined` and the command-line option `-warnings notinlined.`

PowerPC Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.152 shows the **Optimization** panel.

Figure 3.152 Tool Settings — PowerPC Compiler > Optimization



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Manage Configurations…
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Optimization Level: 0

Speed vs. Size: Speed

Inlining: Smart

☐ Bottom-up Inlining

Table 3.141 lists and defines each option of the **Optimization** panel.

Table 3.141 Tool Settings — PowerPC Compiler > Optimization Options

Option	Description
Optimization Level (-opt)	Specify the optimizations that you want the compiler to apply to the generated object code: • **0—**Disable optimizations. This setting is equivalent to specifying the `-O0` command-line option. The compiler generates unoptimized, linear assembly-language code. • **1—**The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the `-O1` command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • **2—**The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the `-O2` command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • **3—**The compiler performs all the level **2** optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the `-O3` command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level **2** optimizations.
Speed Vs Size	Use to specify an **Optimization Level** greater than **0**. • **Speed—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a faster execution speed, as opposed to a smaller executable code size. • **Size—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the `-Os` command-line option.

Table 3.141 Tool Settings — PowerPC Compiler > Optimization Options (*continued*)

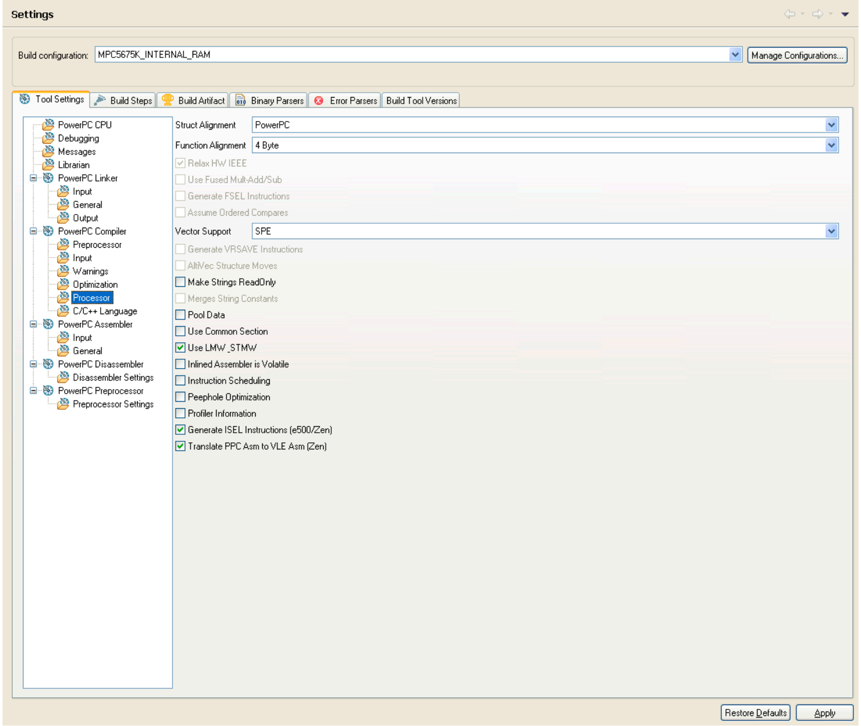
Option	Description
Inlining	Enables inline expansion. If there is a #pragma INLINE before a function definition, all calls of this function are replaced by the code of this function, if possible. Using the -Oi=c0 option switches off inlining. Functions marked with the #pragma INLINE are still inlined. To disable inlining, use the -Oi=OFF option.
Bottom-up Inlining	Check to control the bottom-up function inlining method. When active, the compiler inlines function code starting with the last function in the chain of functions calls, to the first one.

PowerPC Compiler > Processor

Use this panel to specify processor behavior. You can specify the file paths and define macros.

Figure 3.153 shows the **Processor** panel.

Figure 3.153 Tool Settings — PowerPC Compiler > Processor


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Manage Configurations...
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Struct Alignment: PowerPC

Function Alignment: 4 Byte

☑ Relax HW IEEE
☐ Use Fused Mult-Add/Sub
☐ Generate FSEL Instructions
☐ Assume Ordered Compares

Vector Support: SPE

☐ Generate VRSAVE Instructions
☐ AltiVec Structure Moves
☐ Make Strings ReadOnly
☐ Merges String Constants
☐ Pool Data
☐ Use Common Section
☑ Use LMW_STMW
☐ Inlined Assembler is Volatile
☐ Instruction Scheduling
☐ Peephole Optimization
☐ Profiler Information
☑ Generate ISEL Instructions (e500/Zen)
☑ Translate PPC Asm to VLE Asm (Zen)

Table 3.142 lists and defines each option of the **Processor** panel.

Table 3.142 Tool Settings — PowerPC Compiler > Processor

Option	Description
Struct Alignment	The **Struct Alignment** drop-down list has the default selection PowerPC. To conform with the PowerPC EABI and interoperate with third-party object code, this setting should remain PowerPC. Other settings may lead to reduced performance or alignment violation exceptions. If you choose another setting for Struct Alignment, your code may not work correctly.
Function Alignment	If your board has hardware capable of fetching multiple instructions at a time, you may achieve slightly better performance by aligning functions to the width of the fetch. Use the **Function Alignment** drop-down list to select alignments from 4 bytes (the default) to 128 bytes. These selections corresponds to #pragma function_align.
Relax HW IEEE	The **Relax HW IEEE** checkbox is available only if you select Hardware from the **Floating Point** drop-down list. Check the The **Relax HW IEEE** checkbox to have the compiler generate faster code by ignoring certain strict requirements of the IEEE floating-point standard. These requirements are controlled by the options: • Use Fused Multi-Add/Sub • Generate FSEL Instruction • Assume Ordered Compares
Use Fused Mult-Add/Sub	Check to generate PowerPC Fused Multi-Add/Sub instructions, which result in smaller and faster floating-point code. This may generate unexpected results because of the greater precision of the intermediate values. The generated results are slightly more accurate than those specified by IEEE because of an extra rounding bit between the multiply and the add/subtract.

Table 3.142 Tool Settings — PowerPC Compiler > Processor (*continued*)

Option	Description
Generate FSEL Instructions	Check to generate the faster executing FSEL instruction. The FSEL option allows the compiler to optimize the pattern `x = (condition ? y : z)`, where `x` and `y` are floating-point values. FSEL is not accurate for denormalized numbers and may have issues related to unordered compares.

Assume Ordered Compares	Check to allow the compiler to ignore issues with unordered numbers, such as NAN, while comparing floating-point values. In strict IEEE mode, any comparison against NAN, except not-equal-to, returns false.
	This optimization ignores this provision, thus allowing the following conversion:
	`if (a <= b)`
	`to`
	`if (b > a)`

Vector Support	Several processors support vector instructions. If you want to allow vector instructions for your processor, select a vector type that your processor supports from the **Vector Support** drop-down list. If you select the **Altivec** option from the **Vector Support** drop-down list, additional options appear.

Table 3.142 Tool Settings — PowerPC Compiler > Processor (*continued*)

Option	Description
Generate VRSAVE Instructions	Check the **Generate VRSAVE Instructions** checkbox only when developing for a real-time operating system that supports AltiVec. Checking the **Generate VRSAVE Instructions** checkbox tells the CodeWarrior software to generate instructions to save and restore these vector-register-related values. The VRSAVE register indicates to the operating system which vector registers to save and reload when a context switch happens. The bits of the VRSAVE register that correspond to the number of each affected vector register are set to 1. When a function call happens, the value of the VRSAVE register is saved as a part of the stack frame called the vrsave word. In addition, the function saves the values of any non-volatile vector registers in the stack frame as well, in an area called the vector register save area, before changing the values in any of those registers.
AltiVec Structure Moves	Check if you want the CodeWarrior software to use Altivec instructions when the compiler copies a structure.
Make Strings Read Only	Check to store string constants in the read-only .rodata section. Clear to store string constants in the ELF-file data section. The **Make Strings Read Only** checkbox corresponds to #pragma readonly_strings.
Merges String Constants	Check to have the compiler pool strings together from a given file. Clear to let the compiler treat each string as an individual string. The linker can deadstrip unused individual.

Table 3.142 Tool Settings — PowerPC Compiler > Processor (*continued*)

Option	**Description**
Pool Data	Check to instruct the compiler to organize some of the data in the large data sections of .data, .bss, and .rodata so that the program can access it more quickly. This option only affects data that is actually defined in the current source file; it does not affect external declarations or any small data. The linker is normally aggressive in stripping unused data and functions from the C and C++ files in your project. However, the linker cannot strip any large data that has been pooled. If your program uses tentative data, you get a warning that you need to force the tentative data into the common section.
Use Common Section	Check to have the compiler place global uninitialized data in the common section. This section is similar to a FORTRAN Common Block. If the linker finds two or more variables with the same name and at least one of them is in a common section, those variables share the same storage address. If this checkbox is cleared, two variables with the same name generate a link error. The compiler never places small data, pooled data, or variables declared static in the common section.
Use LMW_STMW	Check to have the compiler to use LMW/ STMW instructions in the prologue and epilogue of a function when appropriate to store and restore volatile registers.
Inlined Assembler is Volatile	Check to have the compiler treat all asm blocks (including inline asm blocks) as if the volatile keyword was present. This prevents the asm block from being optimized. You can use the .nonvolatile directive to selectively enable optimization on asm blocks, as required.

Table 3.142 Tool Settings — PowerPC Compiler > Processor (*continued*)

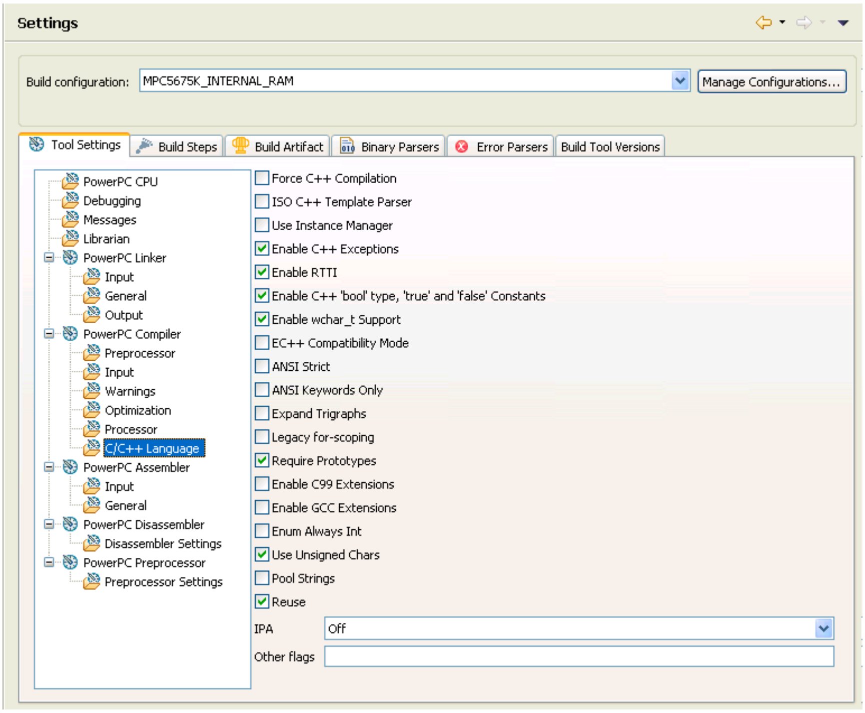
Option	Description
Instruction Scheduling	Check to optimize the scheduling of instructions for the specific processor you are targeting (determined by which processor is selected in the **Processor** drop-down list.) Enabling the **Instruction Scheduling** checkbox can make source-level debugging more difficult because the source code may not correspond to the execution order of the underlying instructions. It is sometimes helpful to clear this checkbox when debugging, and then check it again once you have finished the bulk of your debugging.
Peephole Optimization	Check to have the compiler perform peephole optimizations. Peephole optimizations are small local optimizations that can reduce several instructions into one target instruction, eliminate some compare instructions, and improve branch sequences. This checkbox corresponds to `#pragma peephole`.
Profiler Information	Check to generate special object code during runtime to collect information for a code profiler. This checkbox corresponds to #pragma profile.
Generate ISEL Instructions (e500/Zen)	Check to have the compiler to emit ISEL instructions. This instruction can improve performance by reducing conditional branching.
Translate PPC Asm to VLE Asm (Zen)	Check to have the compiler to translate the classic PPC inline assembly instructions to the VLE inline assembler for the Zen processor.

PowerPC Compiler > C/C++ Language

Use this panel direct the PowerPC compiler to apply specific processing modes to the language source code. You can compile source files with just one collection at a time. To compile source files with multiple collections, you must compile the source code sequentially. After each compile iteration change the collection of settings that the PowerPC compiler uses.

Figure 3.154 shows the **C/C++ Language** panel.

Figure 3.154 Tool Settings — PowerPC Compiler > C/C++ Language


Settings
Build configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings

Build configuration: MPC5675K_INTERNAL_RAM

- ☐ Force C++ Compilation
- ☐ ISO C++ Template Parser
- ☐ Use Instance Manager
- ☑ Enable C++ Exceptions
- ☑ Enable RTTI
- ☑ Enable C++ 'bool' type, 'true' and 'false' Constants
- ☑ Enable wchar_t Support
- ☐ EC++ Compatibility Mode
- ☐ ANSI Strict
- ☐ ANSI Keywords Only
- ☐ Expand Trigraphs
- ☐ Legacy for-scoping
- ☑ Require Prototypes
- ☐ Enable C99 Extensions
- ☐ Enable GCC Extensions
- ☐ Enum Always Int
- ☑ Use Unsigned Chars
- ☐ Pool Strings
- ☑ Reuse

IPA: Off

Other flags:

Table 3.143 lists and defines each option of the **C/C++ Language** panel.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language

Option	Description
Force C++ Compilation	Check to translate all C source files as C++ source code. Clear to use the filename's extension to determine whether to use the C or C++ compiler. The entries in the IDE's File Mappings settings panel specify the suffixes that the compiler assigns to each compiler. This checkbox corresponds to the `pragma cplusplus` and the command-line option -lang `c++`.
ISO C++ Template Parser	Check to follow the ISO/IEC 14882-1998 standard for C++ to translate templates, enforcing more careful use of the typename and template keywords. The compiler also follows stricter rules for resolving names during declaration and instantiation. Clear if you do not want the C+++ compiler expect template source code to follow the ISO C++ standard as closely. This checkbox corresponds to the pragma `parse_func_templ` and the command-line option `-iso_templates`
Use Instance Manager	Check to reduce compile time by generating any instance of a C++ template (or non-inlined inline) function only once. Clear to generate a new instance of a template or non-inlined function each time it appears in source code. This checkbox corresponds to control where the instance database is stored using `#pragma instmgr_file` and command-line option `-instmgr`.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language (*continued*)

Option	Description
Enable C++ Exceptions	Check to generate executable code for C++ exceptions specially when you use the try, throw, and catch statements specified in the ISO/IEC 14882-1998 C++ standard. Clear to generate smaller, faster executable code. The checkbox corresponds to the pragma `exceptions` and the command-line option `-cpp_exceptions`.
Enable RTTI	Check to use of the C++ runtime type information (RTTI) capabilities, including the dynamic_cast and typeid operators. Clear to let the compiler generate smaller, faster object code but do not allow runtime type information operations. The checkbox corresponds to the pragma `RTTI` and the command-line option `-RTTI`.
Enable C++ 'bool' type, 'true' and 'false' Contants	Check to let the C++ compiler recognize the bool type and its true and false values specified in the ISO/IEC 14882-1998 C++ standard. Clear if you do not want the compiler dto recognize this type or its values. The checkbox corresponds to the pragma `bool` and the command-line option `-bool`.
Enable wchar_t Support	Check to let the C++ compiler recognize the `wchar_t` data type specified in the ISO/IEC 14882-1998 C++ standard. Clear if you do not want the compiler to recognize this type or when compiling source code that defines its own wchar_t type. The checkbox corresponds to the pragma `wchar_type` and the command-line option `-wchar_t`.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language (*continued*)

Option	Description
EC++ Compatibility Mode	Check if you expect C++ source code files to contain Embedded C++ source code. Clear if the compiler expects regular C++ source code in C++ source files. The checkbox corresponds to the pragma `ecplusplus` and the command-line option `-dialect ec++`.
ANSI Strict	Check if you want the compiler to only recognize source code that conforms to the ISO/IEC 9899-1990 standard for C. Clear if you want the compiler recognize several CodeWarrior extensions to the C language. The checkbox corresponds to the pragma `ANSI_strict` and the command-line option `-ansi strict`.
ANSI Keywords Only	Check to generate an error message for all non-standard keywords. If you must write source code that strictly adheres to the ISO standard, enable this setting. Clear if you want the compiler to recognize only these non-standard keywords: `far`, `inline`, `__inline__`, `__inline`, and `pascal`. The checkbox corresponds to the pragma `only_std_keywords` and the command-line option `-stdkeywords`.
Expand Trigraphs	Check to let the compiler recognize trigraph sequences (ISO/IEC 9899-1990 C, §5.2.1.1). Clear to ignore trigraph characters. Many common character constants look like trigraph sequences, and this extension lets you use them without including escape characters. The checkbox corresponds to the pragma `trigraphs` and the command-line option `-trigraphs`.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language (*continued*)

Option	Description
Legacy for-scoping	Check to generate an error message when the compiler encounters a variable scope usage that the ISO/IEC 14882-1998 C++ standard disallows. Clear to let the scope rules specified in ARM. The checkbox corresponds to the pragma `require_prototypes` and the command-line option `-requireprotos`.
Require Prototypes	Check to enforce the requirement of function prototypes. The compiler generates an error message if you define a previously referenced function that does not have a prototype. If you define the function before it is referenced but do not give it a prototype, this setting causes the compiler to issue a warning message. Clear if you do not require prototypes. The checkbox corresponds to the pragma `require_prototypes` and the command-line option `-requireprotos`.
Enable C99 Extensions	Check to let the compiler recognize ISO/IEC 9899-1999 ("C99") language features. Clear if you want the compiler to recognize only ISO/IEC 9899-1990 ("C90") language features. The checkbox corresponds to the pragma `gcc_extensions` and the command-line option `-gcc_extensions`.
Enable GCC Extensions	Check to recognize language features of the GNU Compiler Collection (GCC) C compiler that are supported by CodeWarrior compilers. Clear if you do not want the compiler to recognize GCC extensions. The checkbox corresponds to the pragma `gcc_extensions` and the command-line option `-gcc_extensions`.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language (*continued*)

Option	Description
Enum Always Int	Check to use signed integers to represent enumerated constants. Clear to use smallest possible integer type to represent enumerated constants. The checkbox corresponds to the pragma `enumsalwaysint` and the command-line option `-enum`.
Use Unsigned Chars	Check to treat char declarations as unsigned char declarations. Clear to treat char declarations as signed char declarations. The checkbox corresponds to the pragma `unsigned_char` and the command-line option `-char unsigned`.
Pool Strings	Check to collect all string constants into a single data section in the object code it generates. Clear to create a unique section for each string constant. The checkbox corresponds to the pragma `pool_strings` and the command-line option `-strings pool`.
Reuse	Check to store only one copy of identical string literals. Clear to store each string literal separately. The checkbox corresponds to the opposite of the pragma `dont_reuse_strings` and the command-line option `-string reuse`.

Table 3.143 Tool Settings — PowerPC Compiler > C/C++ Language (*continued*)

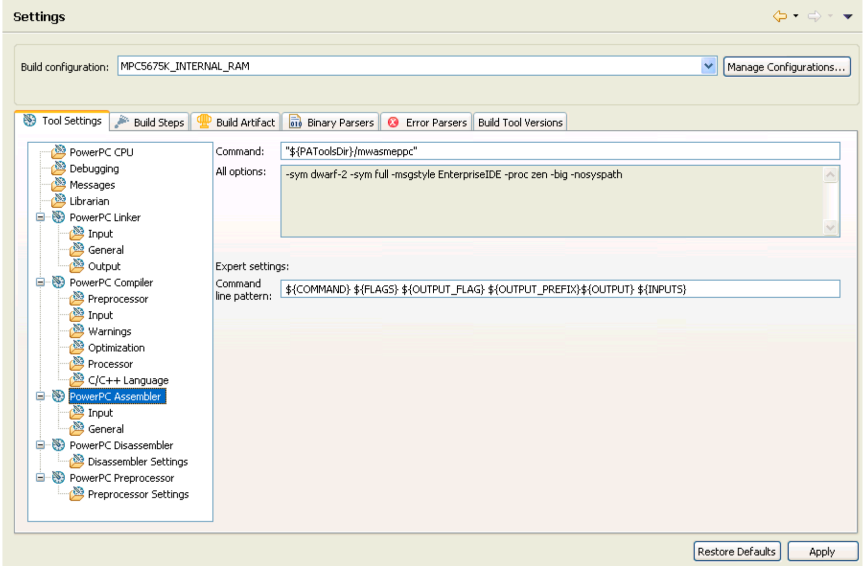
Option	Description
IPA	Specifies the Interprocedural Analysis (IPA) policy. • **Off** — No interprocedural analysis, but still performs function-level optimization. Equivalent to the "no deferred inlining" compilation policy of older compilers. • **File** — Completely parse each translation unit before generating any code or data. Equivalent to the "deferred inlining" option of older compilers. Also performs an early dead code and dead data analysis in this mode. Objects with unreferenced internal linkages will be dead-stripped in the compiler rather than in the linker. • **Program** — Completely parse the entire program before optimizing and generating code, providing many optimization benefits. For example, the compiler can auto-inline functions that are defined in another translation unit. The checkbox corresponds to the command line option `-ipa`.
Other flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

PowerPC Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler. Additionally, the Assembler tree control includes the general and include file search path settings.

Figure 3.155 shows the **Assembler** settings.

Figure 3.155 Tool Settings — PowerPC Assembler



Build configuration: MPC5675K_INTERNAL_RAM

Command: "${PAToolsDir}/mwasmeppc"

All options: -sym dwarf-2 -sym full -msgstyle EnterpriseIDE -proc zen -big -nosyspath

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.144 lists and defines each option of the **PowerPC Assembler** panel.

Table 3.144 Tool Settings — PowerPC Assembler

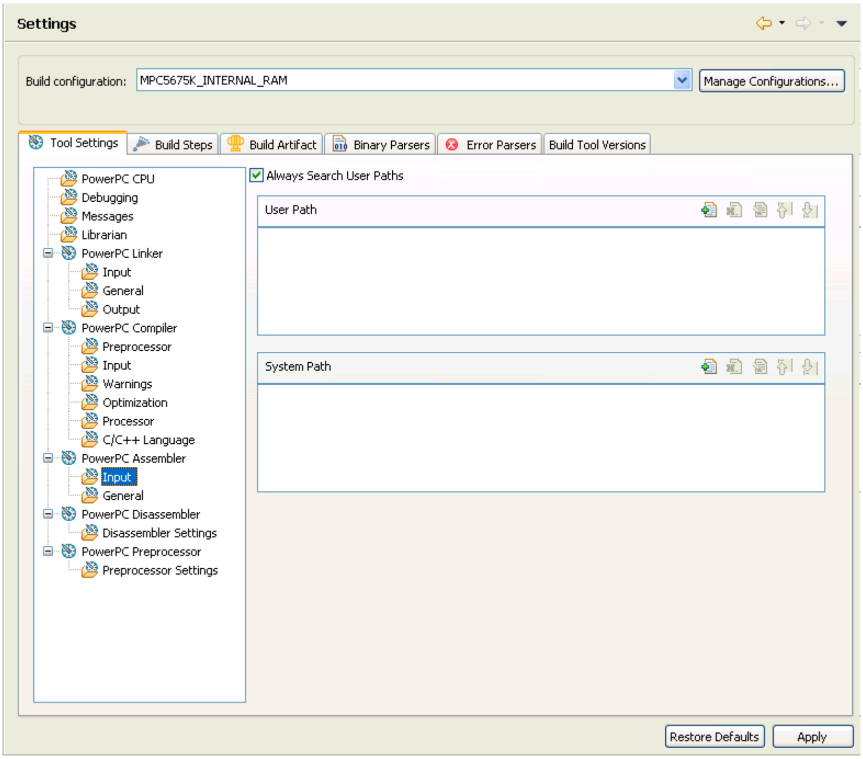
Option	Description
Command	Shows the location of the assembler executable file. Default value is `"${PAToolsDir}/mwasmeppc"`
All options	Shows the actual command line the assembler will be called with.
Expert settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

PowerPC Assembler > Input

Use this panel to specify additional files the **PowerPC Assembler** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.156 shows the **Input** panel.

Figure 3.156 Tool Settings — PowerPC Assembler > Input



Build configuration: MPC5675K_INTERNAL_RAM

☑ Always Search User Paths

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
User Path
System Path
Restore Defaults
Apply

Table 3.145 lists and describes the input options for PowerPC assembler.

Table 3.145 Tool Settings — PowerPC Assembler > Input

Option	Description
Always Search User Paths	CodeWarrior searches only the system path when looking for a file included with #include <…>. To have it searches user path as well, check this box. Note: #include "…" will always search both sets of paths.
User Path (-i)	Lists the available user paths.
System Path	Lists the available system paths.

Table 3.146 lists and describes the toolbar buttons that help work with the user and system search paths.

Table 3.146 Search Paths Toolbar Buttons

Button	Description
✚	**Add —** Click to open the **Add directory path** dialog box (Figure 3.157) and specify the search path.
✖	**Delete —** Click to delete the selected search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.158) and update the selected search path.
⇧	**Move up —** Click to move the selected search path one position higher in the list
⇩	**Move down —** Click to move the selected search path one position lower in the list

Figure 3.157 shows the **Add directory path** dialog box.

Figure 3.157 Add directory path Dialog Box

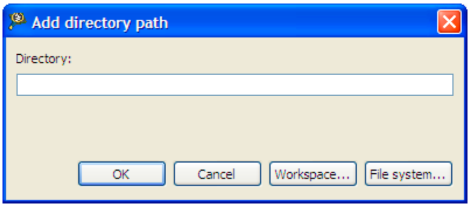
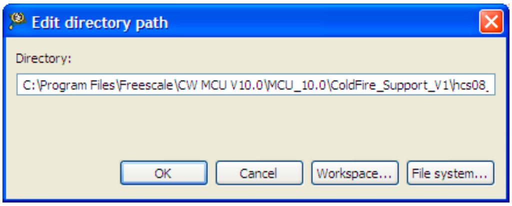

Add directory path
Directory:
OK
Cancel
Workspace...
File system...

Figure 3.158 shows the **Edit directory path** dialog box.

Figure 3.158 Edit directory path Dialog Box


Edit directory path
OK
Cancel
Workspace...
File system...

Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

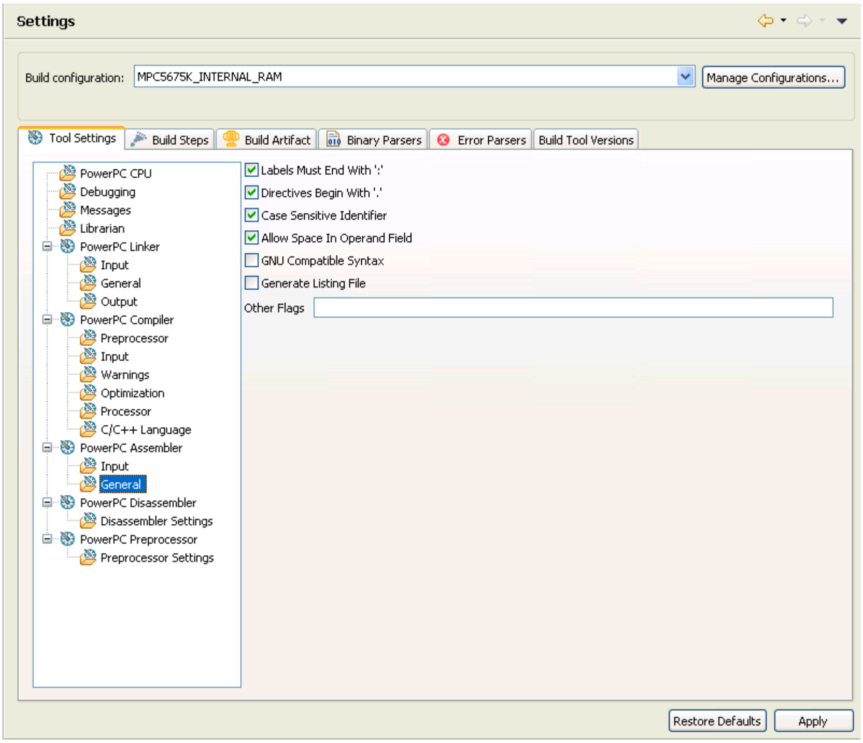
- **OK —** Click to confirm the action and exit the dialog box.
- **Cancel —** Click to cancel the action and exit the dialog box.
- **Workspace —** Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system —** Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

PowerPC Assembler > General

Use this panel to specify the general assembler behavior.

Figure 3.159 shows the **General** panel.

Figure 3.159 Tool Settings — PowerPC Assembler > General



Settings
Build configuration: MPC5675K_INTERNAL_RAM
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

☑ Labels Must End With ':'
☑ Directives Begin With '.'
☑ Case Sensitive Identifier
☑ Allow Space In Operand Field
☐ GNU Compatible Syntax
☐ Generate Listing File
Other Flags

Table 3.147 lists and describes the general assembler options for PowerPC.

Table 3.147 Tool Settings — PowerPC Assembler > General

Option	Description
Labels Must End With ‘:’	Check if the source-file labels must end with colon characters. Clear if the source-file labels need not end with colon characters.
Directives Begin With ‘.’	Check if the assembly directives must begin with period characters. Clear if the assembly directives need not begin with period characters.

Table 3.147 Tool Settings — PowerPC Assembler > General (*continued*)

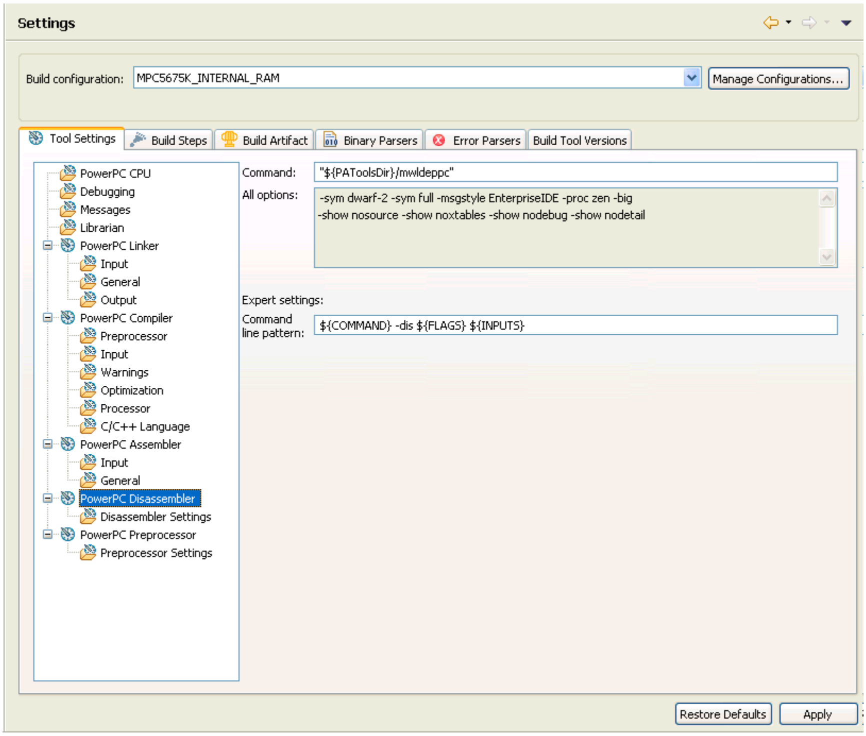
Option	Description
Case Sensitive Identifier	Check if casing matters in identifiers. Clear if the assembler ignores case in identifiers.
Allow Space in Operand Field	Check if spaces are allowed in fields. Clear is spaces are not allowed in fields.
GNU Compatible Syntax	Check if your application does use GNU-compatible syntax. This compatibility allows: • Redefining all equates, regardless if from the .equ or .set directives. • Ignoring the .type directive. • Treating undefined symbols as imported. • Using GNU-compatible arithmetic operators - symbols < and > mean left-shift and right-shift instead of less than and greater than; the symbol ! means bitwise-or-not rather than logical not • Using GNU-compatible precedence rules for operators • Implementing GNU-compatible numeric local labels, from 0 to 9 • Treating numeric constants beginning with 0 as octal • Using semicolons as statement separators • Using a single unbalanced quote for character constants - for example, .byte 'a. Clear to indicate that your application does not use GNU-compatible syntax.
Generate Listing File	Check to let assembler generate a listing file that includes files source, line numbers, relocation information, and macro expansions. Clear if no listing file is specified.
Other Flags	Specify additional command line options for the assembler; type in custom flags that are not otherwise available in the UI.

PowerPC Disassembler

Use this panel to specify the command, options, and expert settings for **PowerPC Disassembler.**

Figure 3.160 shows the **PowerPC Disassembler** page.

Figure 3.160 Tool Settings — PowerPC Disassembler



Build configuration: MPC5675K_INTERNAL_RAM

Command: "${PAToolsDir}/mwldeppc"

All options: -sym dwarf-2 -sym full -msgstyle EnterpriseIDE -proc zen -big -show nosource -show noxtables -show nodebug -show nodetail

Expert settings:

Command line pattern: ${COMMAND} -dis ${FLAGS} ${INPUTS}

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Table 3.148 lists and describes the PowerPC disassembler options.

Table 3.148 Tool Settings — PowerPC Disassembler

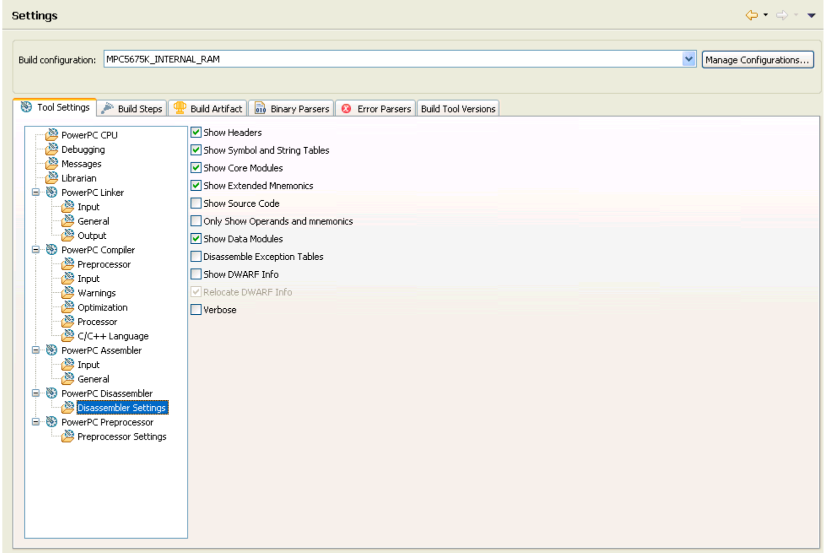
Option	Description
Command	Shows the location of the linker executable file. Default value is `powerpc-linux-gnu-as`
All options	Shows the actual command line the linker will be called with.
Expert settings Command line pattern	Shows the expert settings command line parameters; default is `"${PAGCCINSTALLDIR}/${COMMAND}" ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

PowerPC Disassembler > Disassembler Settings

Use this panel to control how the disassembler formats the listing file, as well as error and warning messages. You can specify verbosity of messages, whether to show headers, core modules, extended mnemonics, addresses, object or source code, ldata modules, exception tables, and debug information.

Figure 3.161 shows the **PowerPC Disassembler** settings.

Figure 3.161 Tool Settings — PowerPC Disassembler Settings


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings

Build configuration: MPC5675K_INTERNAL_RAM

☑ Show Headers
☑ Show Symbol and String Tables
☑ Show Core Modules
☑ Show Extended Mnemonics
☐ Show Source Code
☐ Only Show Operands and mnemonics
☑ Show Data Modules
☐ Disassemble Exception Tables
☐ Show DWARF Info
☑ Relocate DWARF Info
☐ Verbose

Table 3.149 lists and describes the PowerPC disassembler settings.

Table 3.149 Tool Settings — PowerPC Disassembler > Disassembler Settings

Option	Description
Show Headers	Check to display headers in the listing file; disassembler writes listing headers, titles, and subtitles to the listing file
Show Symbol and String Tables	Check to display symbol and string tables directives to the listing file
Show Core Modules	Check to show core modules in the listing file
Show Extended Mnemonics	Check to show the extended mnemonics in the listing file
Show Source Code	Check to show the source code in the listing file
Only Show Operand and Mnemonics	Check this checkbox to have the disassembler list the offset for any functions in the disassembled module.

Table 3.149 Tool Settings — PowerPC Disassembler > Disassembler Settings (*continued*)

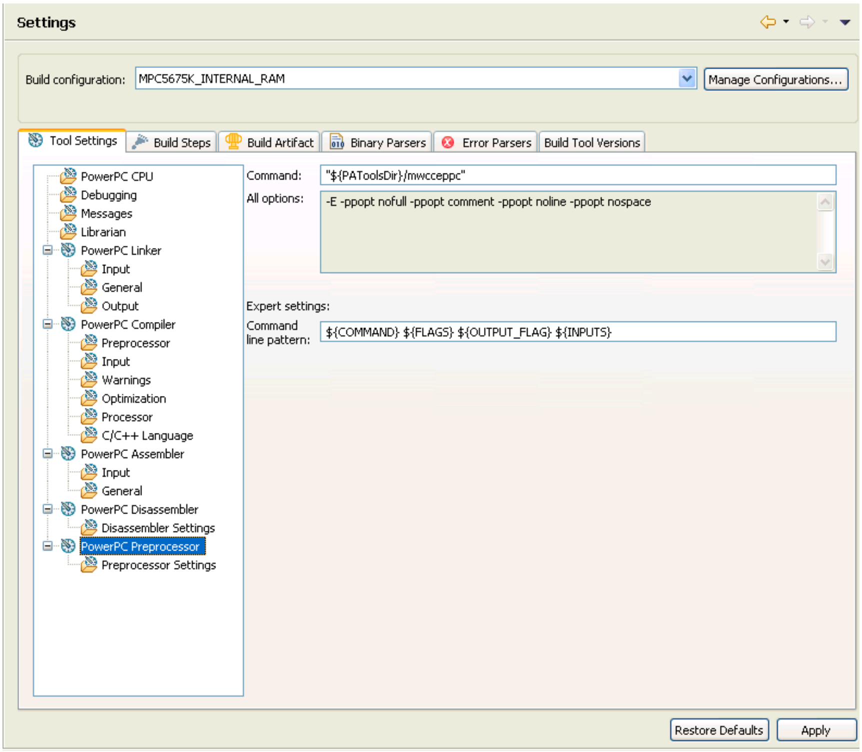
Option	Description
Show Data Modules	Check to show the data modules in the listing file
Disassemble Exception Tables	Check to disassemble exception tables in the listing file
Show DWARF Info	Check to have the disassembler include DWARF symbol information in the disassembled output. Checking this checkbox makes the **Relocate DWARF Info** checkbox available.
Relocate DWARF Info	Check to relocate object and function addresses in the DWARF information.
Verbose	Check to shows each command line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit

PowerPC Preprocessor

Use this panel to specify preprocessor behavior and define macros.

Figure 3.162 shows the **Preprocessor** panel.

Figure 3.162 Tool Settings — PowerPC Preprocessor


Settings
Build configuration:
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Command: "${PAToolsDir}/mwcceppc"

All options: -E -ppopt nofull -ppopt comment -ppopt noline -ppopt nospace

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${INPUTS}

Figure 3.148 lists and describes the preprocessor options for PowerPC.

Table 3.150 Tool Settings — PowerPC Preprocessor

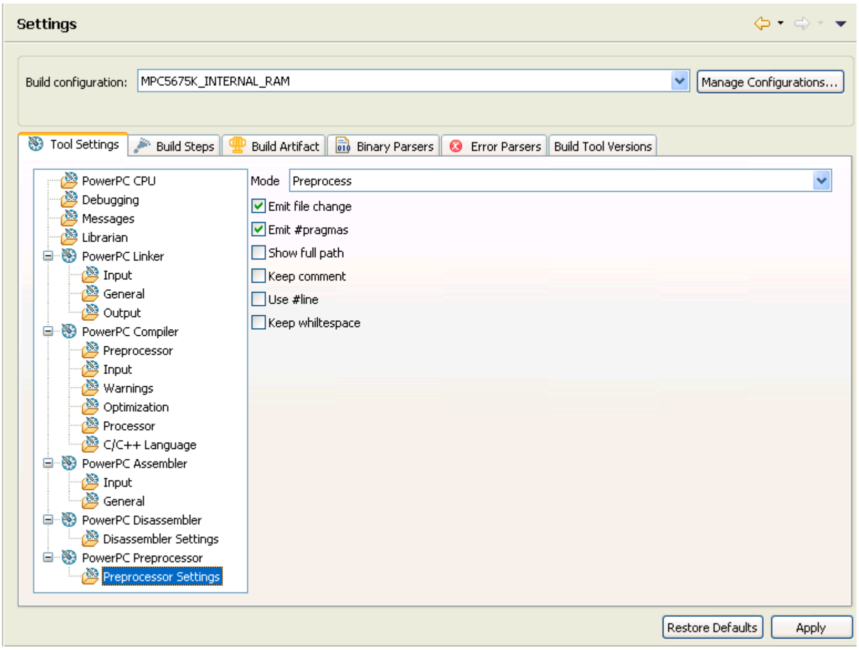
Option	Description
Command	Shows the location of the linker executable file. Default value is `"${PAToolsDir}/mwcceppc"`
All options	Shows the actual command line the linker will be called with.
Expert settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${INPUTS}`

PowerPC Preprocessor > Preprocessor Settings

Use this panel to specify preprocessor behavior.

Figure 3.163 shows the **Preprocessor** panel.

Figure 3.163 Tool Settings — Preprocessor Settings > Preprocessor Settings



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
PowerPC CPU
Debugging
Messages
Librarian
PowerPC Linker
Input
General
Output
PowerPC Compiler
Preprocessor
Input
Warnings
Optimization
Processor
C/C++ Language
PowerPC Assembler
Input
General
PowerPC Disassembler
Disassembler Settings
PowerPC Preprocessor
Preprocessor Settings
Restore Defaults
Apply

Build configuration: MPC5675K_INTERNAL_RAM

Mode: Preprocess

☑ Emit file change
☑ Emit #pragmas
☐ Show full path
☐ Keep comment
☐ Use #line
☐ Keep whitespace

Table 3.151 lists and describes the preprocessor settings options for PowerPC.

Table 3.151 Tool Settings — Preprocessor Settings > Preprocessor Options

Option	Description
Mode	
Emit file change (-ppopt break)	Check to notify file changes (or #line changes) appear in the output.
Emit #pragmas (-ppopt pragma)	Check to show pragma directives in the preprocessor output. Essential for producing reproducible test cases for bug reports.

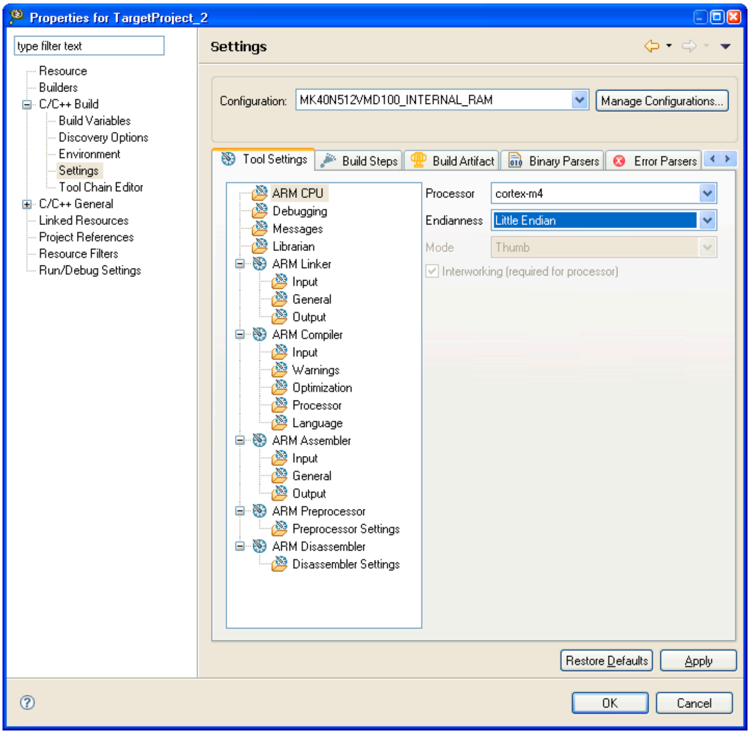
Table 3.151 Tool Settings — Preprocessor Settings > Preprocessor Options (*continued*)

Option	Description
Show full path (-ppopt full)	Check to display file changes in comments (as before) or in #line directives.
Keep comment (-ppopt comment)	Check to display comments in the preprocessor output.
Use #line	
Keep whitespaces	

Build Properties for ARM

The **Properties for <*project*>** window shows the corresponding build properties for ARM CPU project (Figure 3.164).

Figure 3.164 Build Properties - Kinetis


Properties for TargetProject_2
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Restore Defaults
Apply
OK
Cancel

Configuration: MK40N512VMD100_INTERNAL_RAM

Processor: cortex-m4

Endianness: Little Endian

Mode: Thumb

☑ Interworking (required for processor)

Table 3.152 lists the build properties specific to developing software for Kinetis.

The properties that you specify in the **Tool Settings** panels apply to the selected build tool on the **Tool Settings** page of the **Properties for *<project>*** window.

Table 3.152 Build Properties for ARM

Build Tool	Build Properties Panels
ARM CPU	ARM CPU
Debugging	Debugging
Messages	Messages
Librarian	Librarian

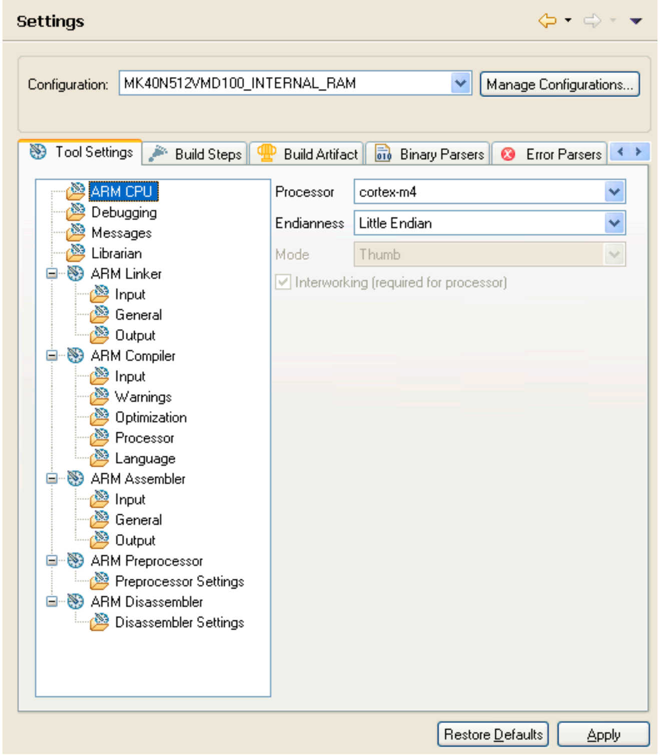
Table 3.152 Build Properties for ARM

Build Tool	Build Properties Panels
ARM Linker	ARM Linker > Input
	ARM Linker > General
	ARM Linker > Output
ARM Compiler	ARM Compiler > Input
	ARM Compiler > Warnings
	ARM Compiler > Optimization
	ARM Compiler > Processor
	ARM Compiler > Language
ARM Assembler	ARM Assembler > Input
	ARM Assembler > General
	ARM Assembler > Output
ARM Preprocessor	ARM Preprocessor > Preprocessor Settings
ARM Disassembler	ARM Disassembler > Disassembler Settings

ARM CPU

Use this panel to specify the CPU type, and the encoding that the architecture uses. The build tools (compiler, linker, and assembler) then use the properties set in this panel to generate CPU-specific code.

Figure 3.165 Tool Settings — ARM CPU


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Restore Defaults
Apply

Configuration: MK40N512VMD100_INTERNAL_RAM

Processor: cortex-m4

Endianness: Little Endian

Mode: Thumb

☑ Interworking (required for processor)

[Table 3.153](#) lists and describes the ARM CPU options.

Table 3.153 Tool Settings — ARM CPU Options

Option	Description
Processor	Lists the processor families supported by the ARM compiler. When you select a processor from this list, the compiler generates code that makes use of any of its hardware features or special instructions. For more detailed information on the features of each processor, refer to its reference manual document.
Endianness	Lists the byte order for the output file.

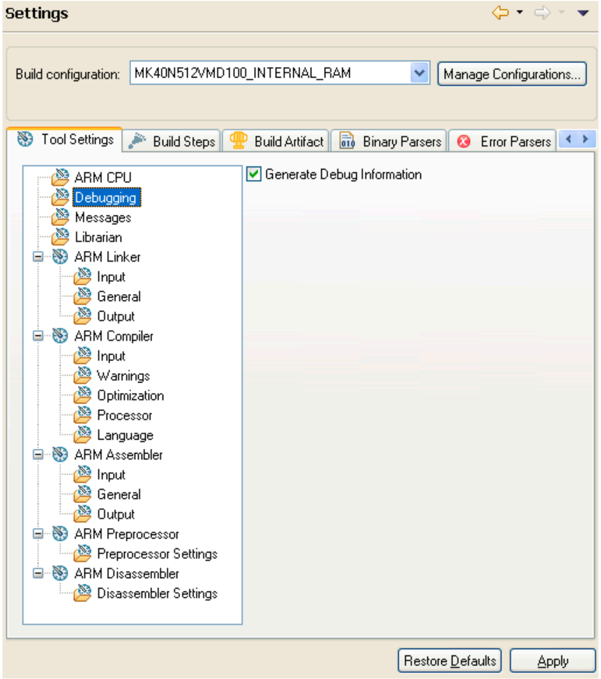
Table 3.153 Tool Settings — ARM CPU Options (*continued*)

Option	Description
Mode	
Interworking (required for processor)	

Debugging

Use this panel to specify the options whether to generate symbolic information for debugging the build target.

Figure 3.166 Tool Settings — Debugging


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Restore Defaults
Apply

Build configuration: MK40N512VMD100_INTERNAL_RAM

☑ Generate Debug Information

Table 3.154 lists and describes the debugging options.

Table 3.154 Tool Settings — Debugging Options

Option	Description
Generate Debug Information	

Messages

Use this panel to specify the options whether to generate symbolic information for debugging the build target (Figure 3.167).

Figure 3.167 Tool Settings — Messages

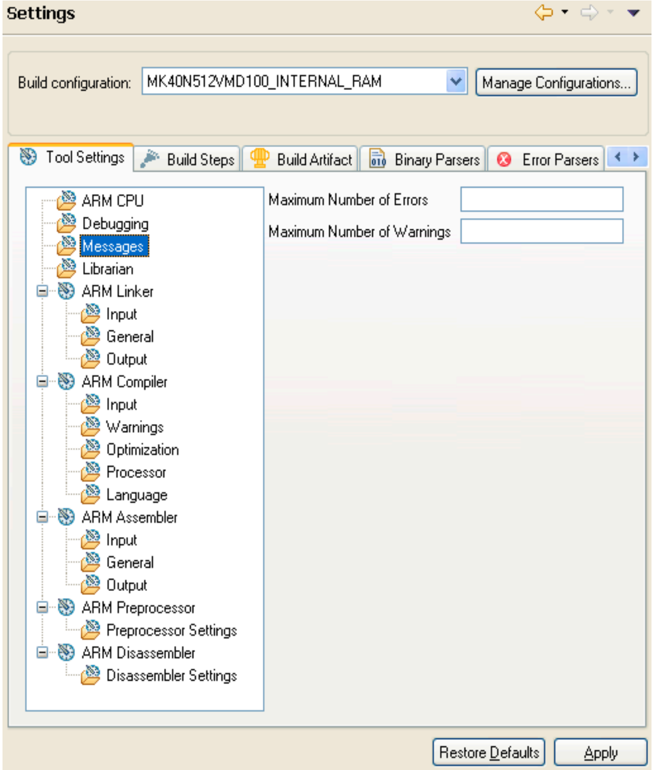

Settings
Build configuration: MK40N512VMD100_INTERNAL_RAM
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Maximum Number of Errors
Maximum Number of Warnings
Restore Defaults
Apply

Table 3.155 lists and describes the message options.

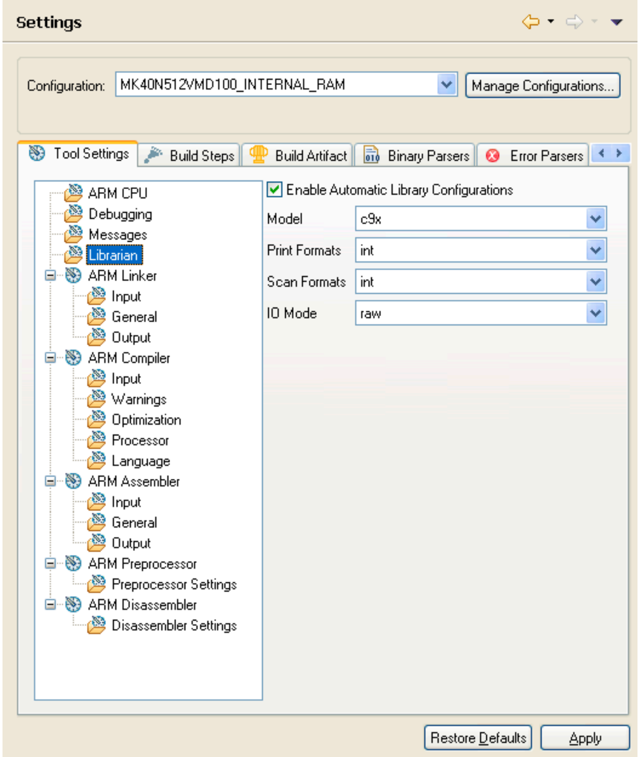
Table 3.155 Tool Settings — Messages Options

Option	Description
Maximum Number of Errors	Specify the number of errors allowed until the application stops processing.
Maximum Number of Warnings	Specify the maximum number of warnings.

Librarian

Use this panel to select options whether the linker will identify standard libraries (Figure 3.168).

Figure 3.168 Tool Settings — Librarian


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Restore Defaults
Apply

Configuration: MK40N512VMD100_INTERNAL_RAM

☑ Enable Automatic Library Configurations

Model: c9x

Print Formats: int

Scan Formats: int

IO Mode: raw

Table 3.156 lists and describes the librarian options.

Table 3.156 Tool Settings — Librarian Options

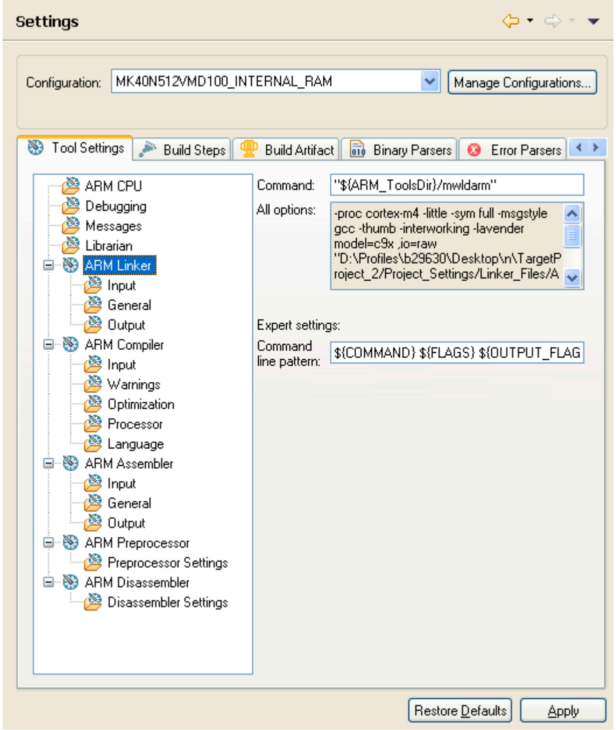
Option	Description
Enable automatic library configurations	Select to let the compiler identify standard libraries.
Model	Select a standard complying or EWL model from the drop-down list. EWL lets you precisely define the I/O operations. EWL drastically reduces the size of executables as you explicitly select the appropriate I/O behavior. Options are: `ewl`, `c9x`, `ewl_c++`, and `c9x_c++`.
Print formats	Select the print formats from the drop-down list. The available options are: `int`, `int_FP`, `int_LL`, and `int_LL_FP`.
Scan formats	Select the scan formats from the drop-down list. The available options are: `int`, `int_FP`, `int_LL`, and `int_LL_FP`.
IO Mode	Select the input-output mode from the drop-down list. The available options are: `raw` and `buffered`.

ARM Linker

Use this panel to specify ARM linker behavior. You can specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the input, general, and output settings.

Figure 3.169 shows the **ARM Linker** settings.

Figure 3.169 Tool Settings — ARM Linker


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Restore Defaults
Apply

Configuration: MK40N512VMD100_INTERNAL_RAM

Command: "${ARM_ToolsDir}/mwldarm"

All options:

```
-proc cortex-m4 -little -sym full -msgstyle
gcc -thumb -interworking -lavender
model=c9x ,io=raw
"D:\Profiles\b29630\Desktop\n\TargetP
roject_2/Project_Settings/Linker_Files/A
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG

Table 3.157 lists and describes the linker options for ARM.

Table 3.157 Tool Settings — ARM Linker Options

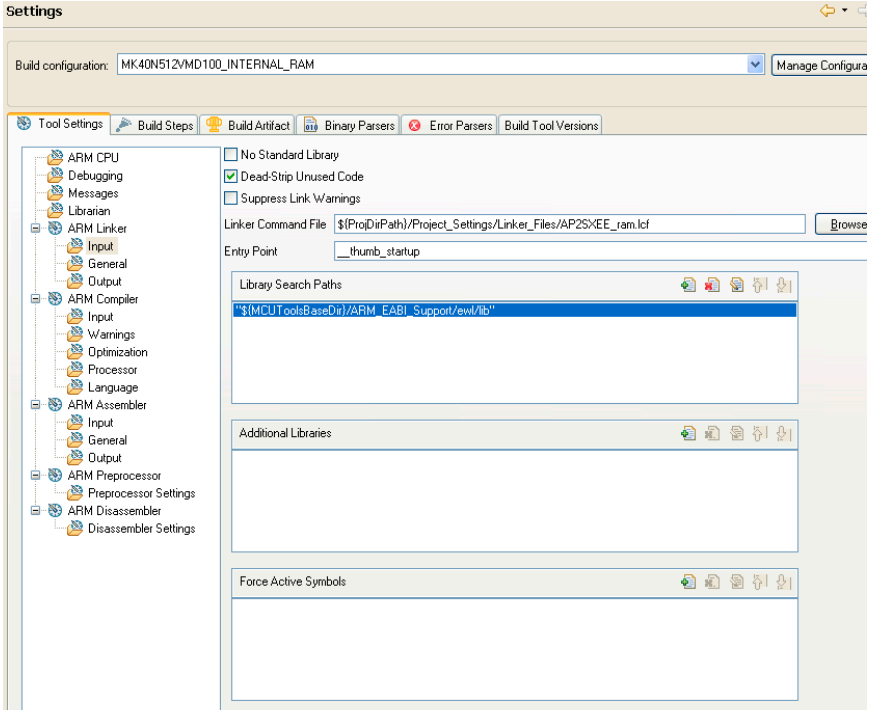
Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${CF_ToolsDir}/mwldarm"`.
All options	Shows the actual command line the ARM linker will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

ARM Linker > Input

Use this panel to specify files the **ARM Linker** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.170 shows the **Input** panel.

Figure 3.170 Tool Settings — ARM Linker > Input



Settings
Manage Configura
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Browse
Library Search Paths
Additional Libraries
Force Active Symbols

Build configuration: MK40N512VMD100_INTERNAL_RAM

☐ No Standard Library
☑ Dead-Strip Unused Code
☐ Suppress Link Warnings

Linker Command File ${ProjDirPath}/Project_Settings/Linker_Files/AP2SXEE_ram.lcf

Entry Point __thumb_startup

Library Search Paths: "${MCUToolsBaseDir}/ARM_EABI_Support/ewl/lib"

Table 3.158 lists and describes the input options for ARM.

Table 3.158 Tool Settings — ARM Linker > Input Options

Option	Description
No Standard Library	
Dead-strip unused code	Determines whether to pool constants from all functions in a file.

Table 3.158 Tool Settings — ARM Linker > Input Options (*continued*)

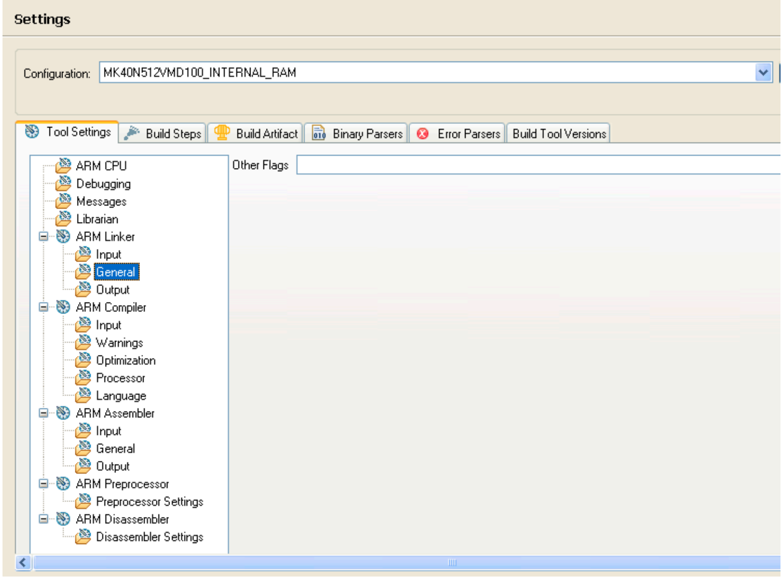
Option	Description
Suppress Link Warnings	Prevents the IDE from displaying linker warning messages.
Linker Command File	Consists of three kinds of segments, which must be in this order: A memory segment, which begins with the MEMORY{} directive Optional closure segments, which begin with the FORCE_ACTIVE{}, KEEP_SECTION{}, or REF_INCLUDE{} directives A sections segment, which begins with the SECTIONS{} directive
Entry Point	Specifies the program starting point: the first function the debugger uses upon program start; default: __thumb_startup. This default function is in file ARM__thumb_startup.c. It sets up the ARM EABI environment before code execution. Its final task is calling main().
Library Search Paths	Specifies the search pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.
Additional Library Files	Specifies the pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click **Workspace** or **File system**, then use the subsequent dialog box to browse to the correct location.
Force Active Symbols	Disables deadstripping for particular symbols, enter the symbol names in the Force Active Symbols text box of the ARM Linker Panel.

ARM Linker > General

Use this panel to specify the general linker behavior.

Figure 3.171 shows the **General** panel.

Figure 3.171 Tool Settings — ARM Linker > General


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Other Flags

Table 3.159 lists and describes the general linker options for ARM.

Table 3.159 Tool Settings — ARM Linker > General Options

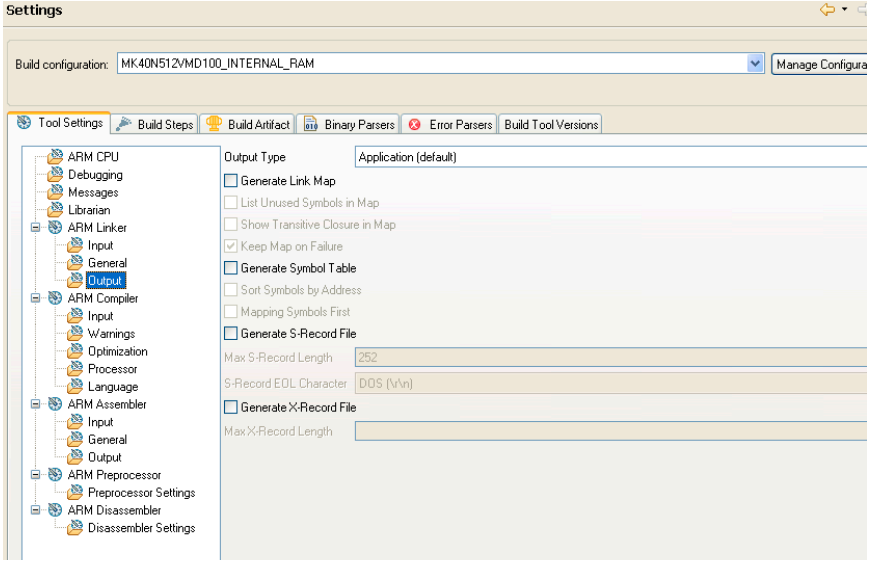
Option	Description
Other Flags	Specify additional command line options for the linker; type in custom flags that are not otherwise available in the UI.

ARM Linker > Output

Use this panel to specify the output settings for the ARM linker.

Figure 3.115 shows the **Output** panel.

Figure 3.172 Tool Settings — ARM Linker > Output


Settings
Manage Configura
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Build configuration: MK40N512VMD100_INTERNAL_RAM

Output Type: Application (default)

☐ Generate Link Map
☐ List Unused Symbols in Map
☐ Show Transitive Closure in Map
☑ Keep Map on Failure
☐ Generate Symbol Table
☐ Sort Symbols by Address
☐ Mapping Symbols First
☐ Generate S-Record File

Max S-Record Length: 252

S-Record EOL Character: DOS (\r\n)

☐ Generate X-Record File

Max X-Record Length:

Table 3.160 lists and describes the output settings for ARM linker.

Table 3.160 Tool Settings — ARM Linker > Output Options

Option	Description
Output Type	Select application as Application (default), Library, or Partial Linking.
Generate Link Map	Check to generate link map.
List Unused Symbols in Map	Check to list unused symbols; appears grayed out if the Generate Link Map checkbox is not checked.
Show Transitive Closure in Map	Check show transitive closure; appears grayed out if the Generate Link Map checkbox is not checked.
Keep Map on Failure	
Generate Symbol Table	Check to generate symbol table.
Sort symbols by address	Check to sort symbols by address; appears grayed out if the Generate Symbol Table checkbox is not checked.

Table 3.160 Tool Settings — ARM Linker > Output Options (*continued*)

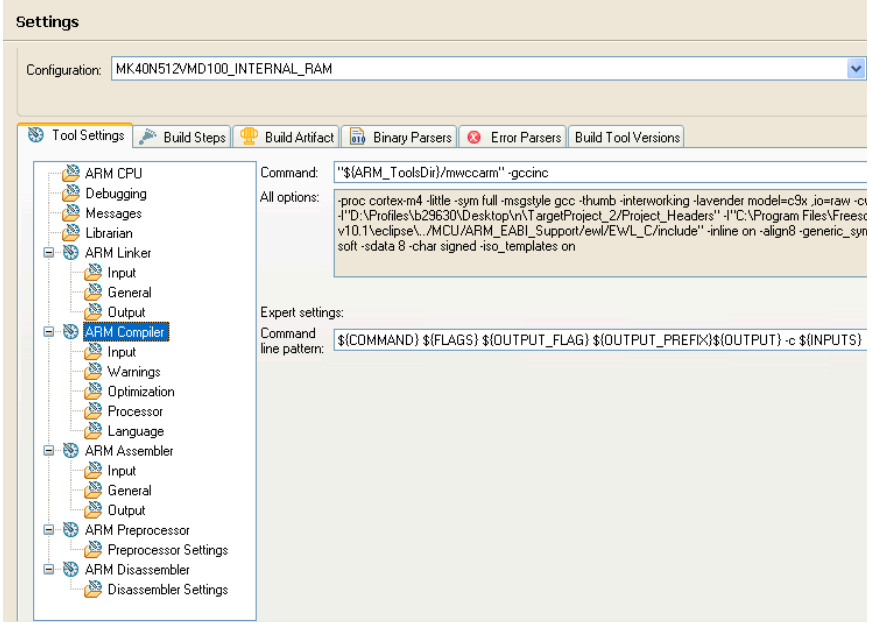
Option	Description
Mapping Symbols First	Check to map symbols first; appears grayed out if the Generate Symbol Table checkbox is not checked.
Generate S-Record File	Check to generate a S-record file.
Max S-Record Length	Specify the maximum length for S-record; appears grayed out if the Generate S-Record File checkbox is not checked. The default value is 252.
S-Record EOL Character	Specify the end-of-line character; appears grayed out if the Generate S-Record File checkbox is not checked. The default value is DOS (\r\n).
Generate X-Record File	Check to generate a X-record file.
Max X-Record Length	Specify the maximum value for X-record; appears grayed out if the Generate X-Record checkbox is not checked.

ARM Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the ARM Compiler tree control includes the general, include file search path settings.

Figure 3.173 shows the **ARM Compiler** settings.

Figure 3.173 Tool Settings — ARM Compiler


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Command: "${ARM_ToolsDir}/mwccarm" -gccinc

All options:

```
-proc cortex-m4 -little -sym full -msgstyle gcc -thumb -interworking -lavender model=c9x ,io=raw -c
-I"D:\Profiles\b29630\Desktop\n\TargetProject_2/Project_Headers" -I"C:\Program Files\Freesc
v10.1\eclipse\../MCU/ARM_EABI_Support/ewl/EWL_C/include" -inline on -align8 -generic_syn
soft -sdata 8 -char signed -iso_templates on
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} -c ${INPUTS}

[Table 3.161](#) lists and describes the compiler options for ARM.

Table 3.161 Tool Settings — ARM Options

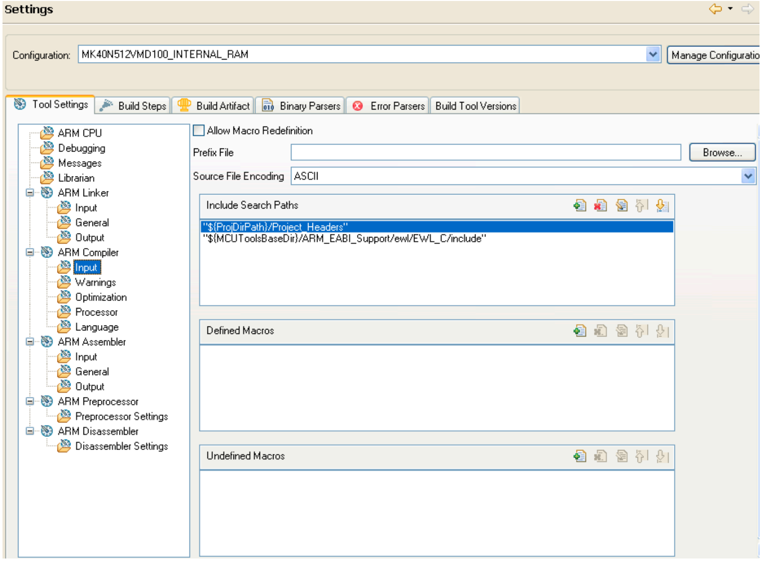
Option	Description
Command	Shows the location of the linker executable file. Default value is: `"${ARM_ToolsDir}/mwccarm" -gccinc`.
All options	Shows the actual command line the ARM compiler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT}-c ${INPUTS}`.

ARM Compiler > Input

Use this panel to specify additional files the **ARM Compiler** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.174 shows the **Input** panel.

Figure 3.174 Tool Settings — ARM Compiler > Input



Settings
Manage Configuratio
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Browse...
Defined Macros
Undefined Macros

Configuration: MK40N512VMD100_INTERNAL_RAM

☐ Allow Macro Redefinition

Prefix File

Source File Encoding ASCII

Include Search Paths

"${ProjDirPath}/Project_Headers"
"${MCUToolsBaseDir}/ARM_EABI_Support/ewl/EWL_C/include"

Table 3.162 lists and describes the input options for ARM compiler.

Table 3.162 Tool Settings — ARM Compiler > Input Options

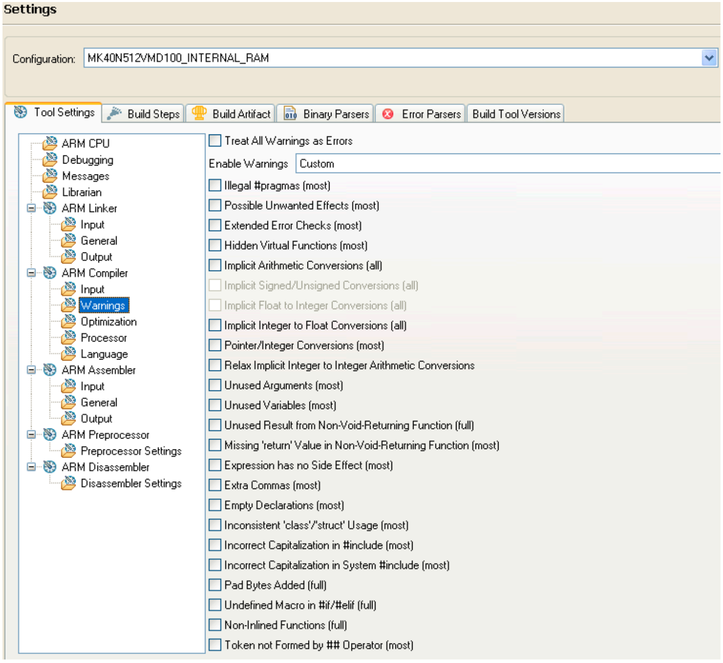
Option	Description
Allow Macro Redefinition	Enables to redefine the macros with the `#define` directive without first undefining them with the `#undef` directive.
Define AEABI Portability	Defines the _AEABI_PORTABILITY_LEVEL to 1. The option ensures that C library dependencies produced by this compiler are ARM EABI compliant. For more information on the ARM EABI and associated compatibility models, the user should consult the ARM, Ltd. website.
Prefix File	Specifies a prefix file that you want the compiler to include at the top of each file.
Source File Encoding	Enables you to specify the default encoding of the source file.
Include Search Paths	Enables you to add new directories to the list of directories where include files are searched.
Defined Macros	Lists the defined command-line macros.
Undefined Macros	Lists the undefined command-line macros.

ARM Compiler > Warnings

Use this panel to control how the ARM compiler formats the listing file, as well as error and warning messages.

Figure 3.175 shows the **Warnings** panel.

Figure 3.175 Tool Settings — ARM Compiler > Warnings


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

☐ Treat All Warnings as Errors

Enable Warnings: Custom

- ☐ Illegal #pragmas (most)
- ☐ Possible Unwanted Effects (most)
- ☐ Extended Error Checks (most)
- ☐ Hidden Virtual Functions (most)
- ☐ Implicit Arithmetic Conversions (all)
- ☐ Implicit Signed/Unsigned Conversions (all)
- ☐ Implicit Float to Integer Conversions (all)
- ☐ Implicit Integer to Float Conversions (all)
- ☐ Pointer/Integer Conversions (most)
- ☐ Relax Implicit Integer to Integer Arithmetic Conversions
- ☐ Unused Arguments (most)
- ☐ Unused Variables (most)
- ☐ Unused Result from Non-Void-Returning Function (full)
- ☐ Missing 'return' Value in Non-Void-Returning Function (most)
- ☐ Expression has no Side Effect (most)
- ☐ Extra Commas (most)
- ☐ Empty Declarations (most)
- ☐ Inconsistent 'class'/'struct' Usage (most)
- ☐ Incorrect Capitalization in #include (most)
- ☐ Incorrect Capitalization in System #include (most)
- ☐ Pad Bytes Added (full)
- ☐ Undefined Macro in #if/#elif (full)
- ☐ Non-Inlined Functions (full)
- ☐ Token not Formed by ## Operator (most)

Table 3.163 lists and describes the warnings options for ARM compiler.

Table 3.163 Tool Settings — ARM Compiler > Warnings Options

Option	Description
Treat All Warnings As Errors	Check to treat all warnings as errors. The compiler will stop if it generates a warning message.
Enable Warnings	Select the level of warnings you want reported from the compiler. Custom lets you to select individual warnings. Other settings select a pre-defined set of warnings.
Illegal #Pragmas	Check to notify the presence of illegal pragmas.
Possible Unwanted Effects	Check to notify most of the possible errors.

Table 3.163 Tool Settings — ARM Compiler > Warnings Options (*continued*)

Option	Description
Extended Error Checks	Check if you want to do an extended error checking.
Hidden virtual functions	Check to generate a warning message if you declare a non-virtual member function that prevents a virtual function, that was defined in a superclass, from being called and is equivalent to pragma `warn_hidevirtua`l and the command-line option `-warnings hidevirtual.`
Implicit Arithmentic Conversions	Check to warn of implict arithmetic conversions.
Implicit Signed/Unsigned Conversion	Check to enable warning of implict conversions between signed and unsigned variables.
Implicit Float to Integer Conversions	Check to warn of implict conversions of a floating-point variable to integer type.
Implicit Integer to Float Conversions	Check to warn of implict conversion of an integer variable to floating-point type.
Pointer/Integer Conversions	Check to enable warnings of conversions between pointer and integers.
Relax Implicit Integer to Interger Arithmetic Conversions	Check to warn when the compiler applies implicit conversion of integer.
Unused Arguments	Check to warn of unused arguments in a function.
Unused Variables	Check to warn of unused variables in the code.
Unused Result From Non-Void-Returning Function	Check to warn of unused result from non-void-returning functions.
Missing ‘return’ value in Non-Void-Returning Function	Check to warn of when a function lacks a return statement.

Table 3.163 Tool Settings — ARM Compiler > Warnings Options (*continued*)

Option	Description
Expression Has No Side Effect	Check to issue a warning message if a source statement does not change the program's state. This is equivalent to the pragma `warn_no_side_effect`, and the command-line option `-warnings unusedexpr`.
Extra Commas	Check to issue a warning message if a list in an enumeration terminates with a comma. The compiler ignores terminating commas in enumerations when compiling source code that conforms to the ISO/IEC 9899-1999 ("C99") standard and is equivalent to pragma `warn_extracomma` and the command-line `option -warnings extracomma`.
Empty Declarations	Check to warn of empty declarations.
Inconsistent ‘class’ / ‘struct’ Usage	Check to warn of inconsistent usage of class or struct.
Incorrect Capitalization in #include	Check to issue a warning message if the name of the file specified in a #include "file" directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps` and the command-line option `-warnings filecaps`.
Incorrect Capitalization in System #Includes	Check to issue a warning message if the name of the file specified in a #include <file> directive uses different letter case from a file on disk and is equivalent to pragma `warn_filenamecaps_system` and the command-line option `-warnings sysfilecaps`.
Pad Bytes Added	Check to issue a warning message when the compiler adjusts the alignment of components in a data structure and is equivalent to pragma `warn_padding` and the command-line option `-warnings padding`.

Table 3.163 Tool Settings — ARM Compiler > Warnings Options (*continued*)

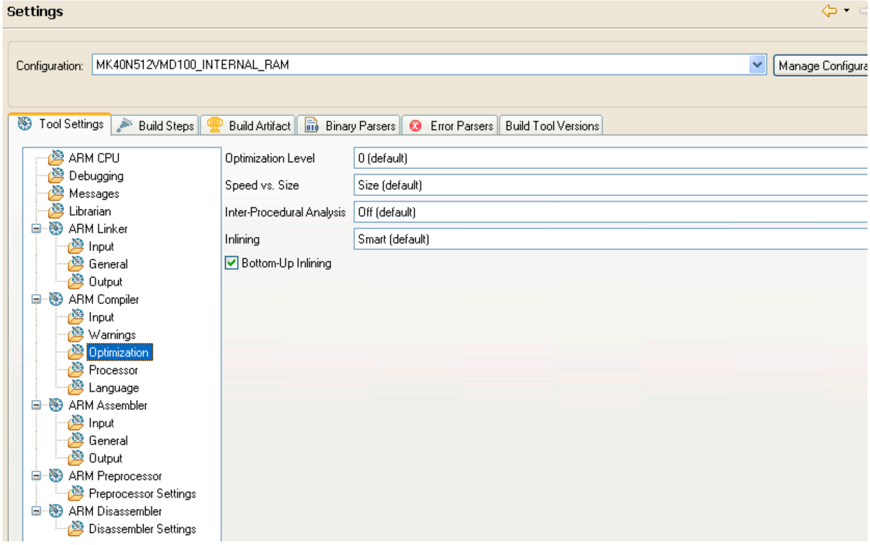
Option	Description
Undefined Macro in #if/#elif	Check to issues a warning message if an undefined macro appears in #if and #elif directives and is equivalent to pragma `warn_undefmacro` and the command-line option `-warnings undefmacro`.
Non-Inlined Functions	Check to issue a warning message if a call to a function defined with the inline, __inline__, or __inline keywords could not be replaced with the function body and is equivalent to pragma `warn_notinlined` and the command-line option `-warnings notinlined`.
Token not formed by ## Operator	Check to enable warnings for the illegal uses of the preprocessor's token concatenation operator (##). It is equivalent to the pragma `warn_illtokenpasting on`.

ARM Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 3.176 shows the **Optimization** panel.

Figure 3.176 Tool Settings — ARM Compiler > Optimization


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configura
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Optimization Level: 0 (default)
Speed vs. Size: Size (default)
Inter-Procedural Analysis: Off (default)
Inlining: Smart (default)
☑ Bottom-Up Inlining

Table 3.164 lists and defines each option of the **Optimization** panel.

Table 3.164 Tool Settings — ARM Compiler > Optimization Options

Option	Description
Optimization Level (-opt)	Specify the optimizations that you want the compiler to apply to the generated object code: **0—**Disable optimizations. This setting is equivalent to specifying the `-O0` command-line option. The compiler generates unoptimized, linear assembly-language code. **1—**The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the `-O1` command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. **2—**The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the `-O2` command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. **3—**The compiler performs all the level **2** optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the `-O3` command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level **2** optimizations. **4—** The compiler performs all the level **3** optimizations. This setting is equivalent to specifying the `-O3` command-line option. At this level, the compiler adds repeated subexpression elimination and loop-invariant code motion.

Table 3.164 Tool Settings — ARM Compiler > Optimization Options (*continued*)

Speed Vs Size	Use to specify an **Optimization Level** greater than **0**. • **Speed—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a faster execution speed, as opposed to a smaller executable code size. • **Size—**The compiler optimizes object code at the specified **Optimization Level** such that the resulting binary file has a smaller executable code size, as opposed to a faster execution speed. This setting is equivalent to specifying the `-Os` command-line option.
Inter-Procedural Analysis	Control whether the compiler views single or multiple source files at compile time. • **Off—**Compiler compiles one file at a time. The functions are displayed in order as they appear in the source file. An object file is created for each source. • **File—**The compiler sees all the functions and data in a translation unit (source file) before code or data is generated. This allows inlining of functions that may not have been possible in -ipa off mode. • **Program—**The compiler sees all the source files of a project before code and data are generated. This allows for cross-module optimizations, including inlining. • **Program-Final—**The compiler sees all the source files of a project before code and data are generated. The object files are passed explicitly to the linker.

Table 3.164 Tool Settings — ARM Compiler > Optimization Options (*continued*)

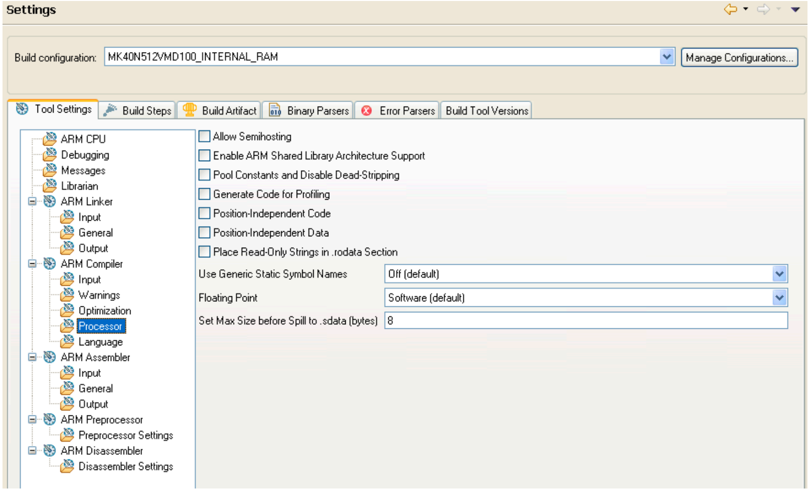
Inlining	Enables inline expansion. If there is a #pragma INLINE before a function definition, all calls of this function are replaced by the code of this function, if possible. Using the -Oi=c0 option switches off inlining. Functions marked with the #pragma INLINE are still inlined. To disable inlining, use the -Oi=OFF option. • **Smart—** Inlines function declared with the inline qualifier • **Auto—** Inlines small function even if they are not declared with the inline qualifier • **Off—** No functions are inlined.
Bottom-up Inlining	Check to control the bottom-up function inlining method. When active, the compiler inlines function code starting with the last function in the chain of functions calls, to the first one.

ARM Compiler > Processor

Use this panel to specify processor behavior. You can specify the file paths and define macros.

Figure 3.177 shows the **Processor** panel.

Figure 3.177 Tool Settings — ARM Compiler > Processor


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Build configuration: MK40N512VMD100_INTERNAL_RAM

☐ Allow Semihosting
☐ Enable ARM Shared Library Architecture Support
☐ Pool Constants and Disable Dead-Stripping
☐ Generate Code for Profiling
☐ Position-Independent Code
☐ Position-Independent Data
☐ Place Read-Only Strings in .rodata Section

Use Generic Static Symbol Names: Off (default)
Floating Point: Software (default)
Set Max Size before Spill to .sdata (bytes): 8

Table 3.165 lists and defines each option of the **Processor** panel.

Table 3.165 Tool Settings — ARM Compiler > Processor Options

Option	Description
Allow Semihosting	This option enables users to ARM semihosting features in their applications (such as console output).
Enable ARM Shared Library Architecture Support	This option is disabled for this product.
Pool Constants and Disable Dead-Stripping	This option combines the literal constant pools of multiple functions in a translation unit. The implication of this is that multiple functions reside in a single ELF section, thus disabling dead-stripping by the linker (each function must reside in a unique ELF section to be dead-stripped).
Generate Code for Profiling	Check to enable the processor generate code for use with a profiling tool. Checking this box corresponds to using the command-line option `-profile`. Clearing this checkbox is equivalent to using the command-line option `-noprofile`.

Table 3.165 Tool Settings — ARM Compiler > Processor Options (*continued*)

Option	Description
Position-Independent Code	Equivalent to -pic, Using PIC frees you from having to commit to loading your code at a particular address in memory. This means that the code can move to different memory locations and still work correctly. The generated code that is the same regardless of its load address.
Position-Independent Data	Equivalent to -pid, Using PID frees you from having to commit to loading your data at a particular address in memory. This means that the data can move to different memory locations and still work correctly.
Place Read-Only Strings in .rodata Section	Instructs the compiler to place string constants into the `.rodata` section.
Use Generic Static Symbol Names	Obfuscates the name of static symbols within binaries (such as libraries) as protective measure against unauthorized persons disassembling the binary. Such a disassembly can reveal the names of static symbols and may expose internal structures and other proprietary details.

Table 3.165 Tool Settings — ARM Compiler > Processor Options (*continued*)

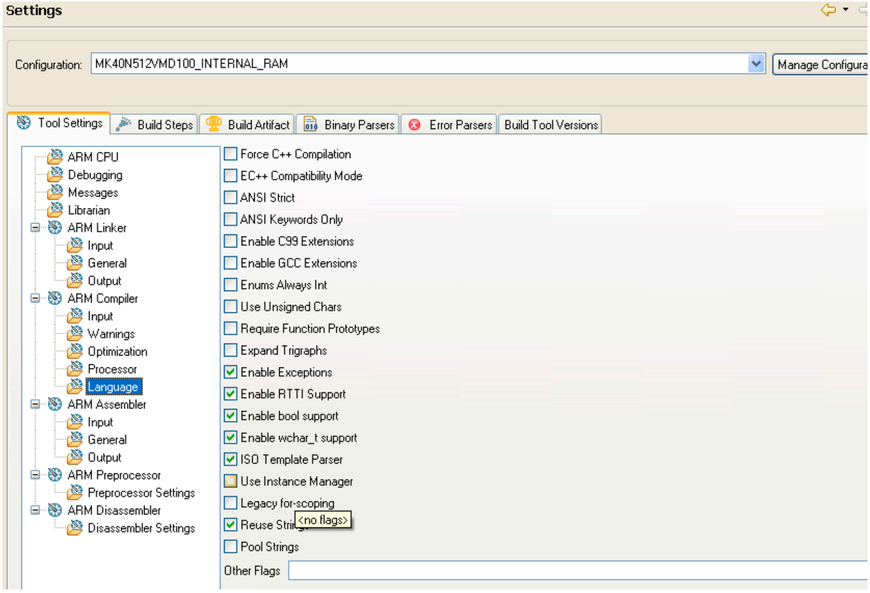
Option	Description
Floating Point	Specifies handling method for floating point operations: • **Software** — C runtime library code emulates floating-point operations. • **Hardware** — Processor hardware performs floating point operations; only appropriate for processors that have floating-point units. • **None** Default: Software For software selection, your project must include the appropriate FP_ARM C runtime library file. Grayed out if your target processor lacks an FPU.
Set Max Size before Spill to .sdata (bytes)	Enter the maximum number of bytes (n) of an object length for which the processor uses a .sdata section.

ARM Compiler > Language

Use this panel direct the ARM compiler to apply specific processing modes to the language source code. You can compile source files with just one collection at a time. To compile source files with multiple collections, you must compile the source code sequentially. After each compile iteration change the collection of settings that the ARM compiler uses.

Figure 3.123 shows the **Language** panel.

Figure 3.178 Tool Settings — ARM Compiler > Language Settings



Settings
Manage Configura
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
<no flags>

Configuration: MK40N512VMD100_INTERNAL_RAM

☐ Force C++ Compilation
☐ EC++ Compatibility Mode
☐ ANSI Strict
☐ ANSI Keywords Only
☐ Enable C99 Extensions
☐ Enable GCC Extensions
☐ Enums Always Int
☐ Use Unsigned Chars
☐ Require Function Prototypes
☐ Expand Trigraphs
☑ Enable Exceptions
☑ Enable RTTI Support
☑ Enable bool support
☑ Enable wchar_t support
☑ ISO Template Parser
☐ Use Instance Manager
☐ Legacy for-scoping
☑ Reuse Strings
☐ Pool Strings
Other Flags

Table 3.166 lists and defines each option of the **Language Settings** panel.

Table 3.166 Tool Settings — ARM Compiler > Language Settings Options

Option	Description
Force C++ Compilation	Check to translate all C source files as C++ source code.
ANSI Strict	Check to enable C compiler operate in strict ANSI mode. In this mode, the compiler strictly applies the rules of the ANSI/ISO specification to all input files. This setting is equivalent to specifying the `- ansi` command-line option. The compiler issues a warning for each ANSI/ISO extension it finds.

Table 3.166 Tool Settings — ARM Compiler > Language Settings Options (*continued*)

Option	**Description**
ANSI Keywords Only	Check to generate an error message for all non-standard keywords (ISO/IEC 9899-1990 C, §6.4.1). If you must write source code that strictly adheres to the ISO standard, enable this setting; is equivalent to pragma `only_std_keywords` and the command-line option `-stdkeywords`.
Enable C99 Extensions	Check to recognize ISO/IEC 9899-1999 ("C99") language features; is equivalent to pragma `c99` and the command-line option `-dialect c99`.
Enable GCC Extensions	Check to recognize language features of the GNU Compiler Collection (GCC) C compiler that are supported by CodeWarrior compilers; is equivalent to pragma `gcc_extensions` and the command-line option `-gcc_extensions.`
Enums Always Int	Check to use signed integers to represent enumerated constants and is equivalent to pragma `enumsalwaysint` and the command-line option `-enum`.
Use Unsigned Chars	Check to treat char declarations as unsigned char declarations and is equivalent to pragma `unsigned_char` and the command-line option `-char unsigned.`
Require Function Prototypes	Check to enforce the requirement of function prototypes. The compiler generates an error message if you define a previously referenced function that does not have a prototype. If you define the function before it is referenced but do not give it a prototype, this setting causes the compiler to issue a warning message.
Expand Trigraphs	Check to recognize trigraph sequences (ISO/IEC 9899-1990 C, §5.2.1.1); is equivalent to **pragma** `trigraphs` **and the command-line option** `-trigraphs.`

Table 3.166 Tool Settings — ARM Compiler > Language Settings Options (*continued*)

Option	Description
Enable Exceptions	Check to generate executable code for C++ exceptions; is equivalent to pragma `exceptions` and the command-line option `-cpp_exceptions`.
Enable RTTI Support	Check to allow the use of the C++ runtime type information (RTTI) capabilities, including the `dynamic_cast` and `typeid` operators; is equivalent to pragma RTTI and the command-line option -RTTI.
Enable bool Support	Check to enable the C++ compiler to recognize the `bool` type and its `true` and `false` values specified in the ISO/IEC 14882-1998 C++ standard.
Enable wchar_t Support	Check to enable C++ compiler recognize the wchar_t data type specified in the ISO/IEC 14882-1998 C++ standard; is equivalent to pragma `wchar_type` and the command-line option `-wchar_t`.
ISO Template Parser	Check to follow the ISO/IEC 14882-1998 standard for C++ to translate templates, enforcing more careful use of the `typename` and `template` keywords. The compiler also follows stricter rules for resolving names during declaration and instantiation.
Use Instance Manager	Check to reduce compile time by generating any instance of a C++ template (or non-inlined inline) function only once.
Legacy for-scoping	Check to generate an error message when the compiler encounters a variable scope usage that the ISO/IEC 14882-1998 C++ standard disallows, but is allowed in the C++ language specified in The Annotated C++ Reference Manual ("ARM"); is equivalent to pragma `ARM_scoping` and the command-line option `-for_scoping`.

Table 3.166 Tool Settings — ARM Compiler > Language Settings Options (*continued*)

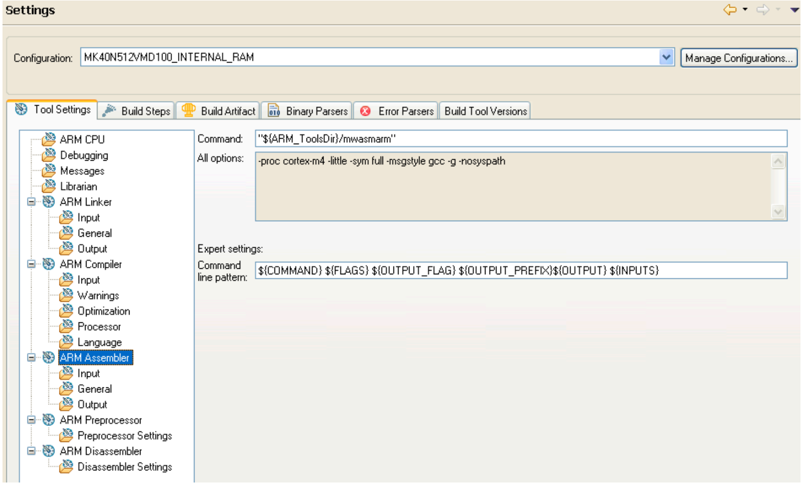
Option	Description
Reuse Strings	Check to store only one copy of identical string literals and is equivalent to opposite of the pragma `dont_reuse_strings` and the command-line `option -string reuse.`
Pool Strings	Check to collect all string constants into a single data section in the object code it generates and is equivalent to pragma `pool_strings` and the command-line `option -strings pool.`
Other flags	Specify additional command line options for the compiler; type in custom flags that are not otherwise available in the UI. **Note**: To enable CodeWarrior MCU V10.x to generate .lst file for each source file in ARM, you need to specify -S in the **Other Flags** option.

ARM Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler. Additionally, the Assembler tree control includes the general and include file search path settings.

Figure 3.179 shows the **Assembler** settings.

Figure 3.179 Tool Settings — ARM Assembler



Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Command: "${ARM_ToolsDir}/mwasmarm"

All options: -proc cortex-m4 -little -sym full -msgstyle gcc -g -nosyspath

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

Table 3.167 lists and defines each option of the **ARM Assembler** panel.

Table 3.167 Tool Settings — ARM Assembler Options

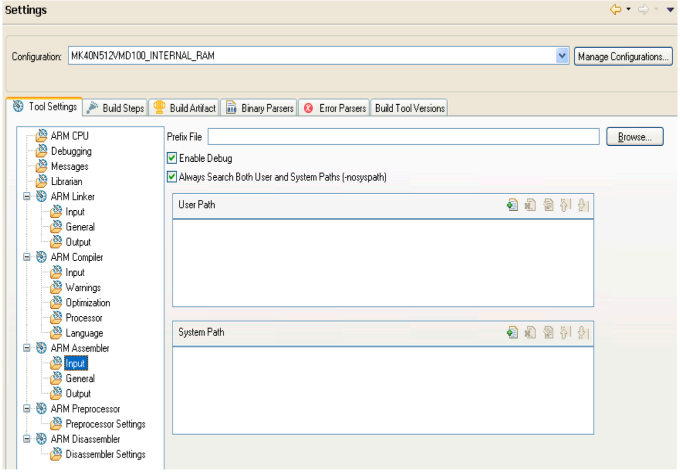
Option	Description
Command	Shows the location of the assembler executable file.
All options	Shows the actual command line the assembler will be called with.
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is `${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`.

ARM Assembler > Input

Use this panel to specify additional files the **ARM Assembler** should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 3.180 shows the **Input** panel.

Figure 3.180 Tool Settings — ARM Assembler > Input



Configuration: MK40N512VMD100_INTERNAL_RAM

Prefix File

☑ Enable Debug

☑ Always Search Both User and System Paths (-nosyspath)

Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings
Browse...
User Path
System Path

Table 3.168 lists and describes the input options for ARM assembler.

Table 3.168 Tool Settings — ARM Compiler > Input Options

Option	Description
Prefix File	Specify a prefix file that you want the compiler to include at the top of each file.
Enable Debug	Check to automatically generate the debug information for the project.
Always Search Both User and System Paths (-nosyspath)	Performs a search of both the user and system paths, treating `#include` statements of the form `#include <xyz>` the same as the form `#include “xyz”`.
User Path	Lists the available user paths.
System Path	Lists the available system paths.

Table 3.169 lists and describes the toolbar buttons that help work with the user and system search paths.

Table 3.169 Search Paths Toolbar Buttons

Button	Description
➕	**Add —** Click to open the **Add directory path** dialog box (Figure 3.181) and specify the search path.
✖	**Delete —** Click to delete the selected search path. To confirm deletion, click **Yes** in the **Confirm Delete** dialog box.
✎	**Edit —** Click to open the **Edit directory path** dialog box (Figure 3.182) and update the selected search path.
⇧	**Move up —** Click to move the selected search path one position higher in the list
⇩	**Move down —** Click to move the selected search path one position lower in the list

Figure 3.181 shows the **Add directory path** dialog box.

Figure 3.181 Add directory path Dialog Box

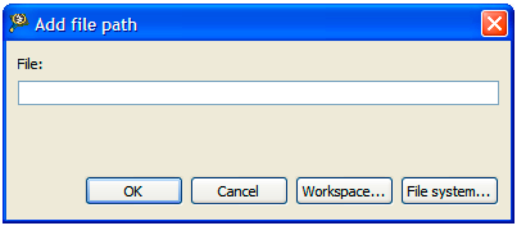
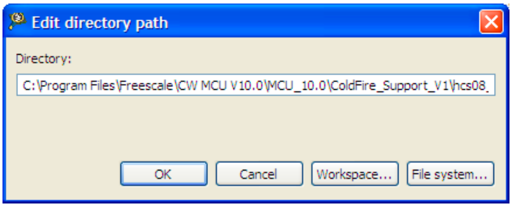


Figure 3.182 shows the **Edit directory path** dialog box.

Figure 3.182 Edit directory path Dialog Box



Directory: C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\ColdFire_Support_V1\hcs08_

The buttons in the **Add directory path and Edit directory path** dialog boxes help work with the object file search paths.

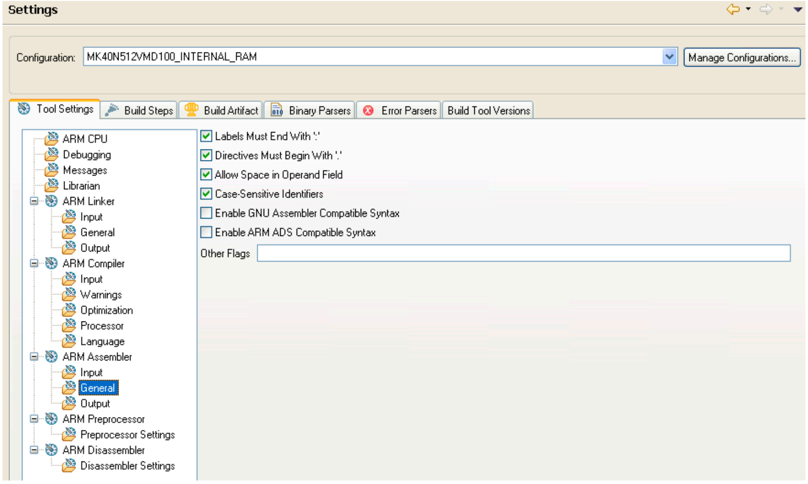
- OK — Click to confirm the action and exit the dialog box.
- **Cancel** — Click to cancel the action and exit the dialog box.
- **Workspace** — Click to display the **Folder Selection** dialog box and specify the object file search path. The resulting path, relative to the workspace, appears in the appropriate list.
- **File system** — Click to display the **Browse for Folder** dialog box and specify the object file search path. The resulting path appears in the appropriate list.

ARM Assembler > General

Use this panel to specify the general assembler behavior.

Figure 3.183 shows the **General** panel.

Figure 3.183 Tool Settings — ARM Assembler > General


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

☑ Labels Must End With ':'
☑ Directives Must Begin With '.'
☑ Allow Space in Operand Field
☑ Case-Sensitive Identifiers
☐ Enable GNU Assembler Compatible Syntax
☐ Enable ARM ADS Compatible Syntax
Other Flags

Table 3.170 lists and describes the general assembler options for ARM.

Table 3.170 Tool Settings — ARM Assembler > General Options

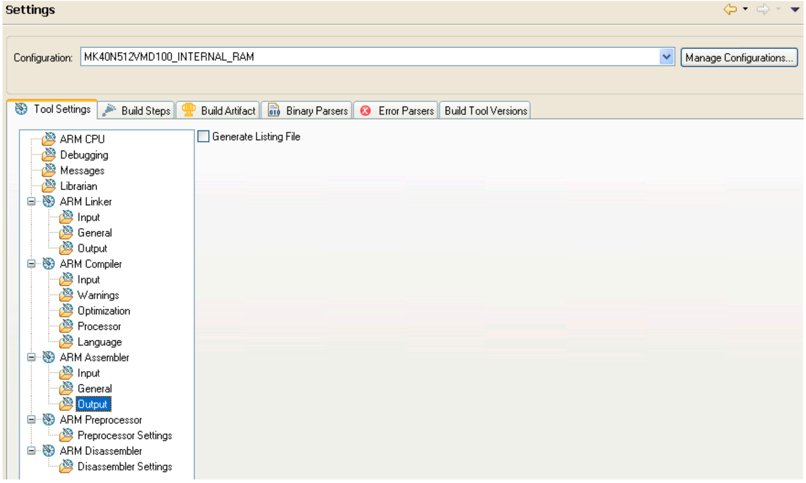
Option	Description
Label Must End With ':'	Clear if system does not require labels to end with colons. By default, the option is checked.
Directives Begin With '.'	Clear if the system does not require directives to start with periods. By default, the option is checked.
Allow Space In Operand Field	Clear to restrict the assembler from adding spaces in operand fields. By default, the option is checked.
Case Sensitive Identifier	Clear to instruct the assembler to ignore case in identifiers. By default, the option is checked.
Enable GNU Assembler Compatible Syntax	Instructs the assembler to accept GNU-style assembly syntax.
Enable ARM ADS Compatible Syntax	Instructs the assembler to accept ARM ADS compatible syntax extensions.
Other Flags	Specify additional command line options for the assembler; type in custom flags that are not otherwise available in the UI.

ARM Assembler > Output

Use this panel to specify the output settings for the ARM assembler.

Figure 3.115 shows the **Output** panel.

Figure 3.184 Tool Settings — ARM Assembler > Output


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

☐ Generate Listing File

Table 3.171 lists and describes the output assembler options for ARM.

Table 3.171 Tool Settings — ARM Assembler > Output Options

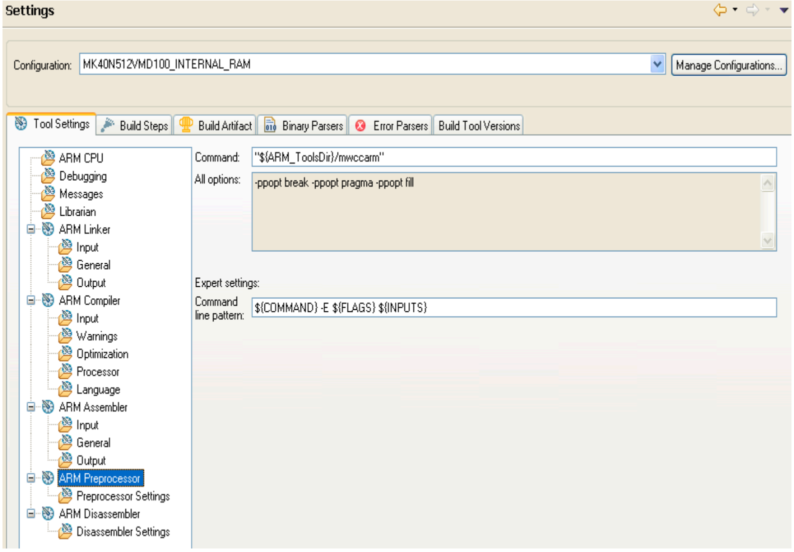
Option	Description
Generate Listing File	Instructs the assembler to generate a disassembly output file. The disassembly output file contains the file source, along with line numbers, relocation information, and macro expansion.

ARM Preprocessor

Use this panel to specify preprocessor behavior and define macros.

Figure 3.185 shows the **Preprocessor** panel.

Figure 3.185 Tool Settings — ARM Preprocessor



Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Command: "${ARM_ToolsDir}/mwccarm"

All options: -ppopt break -ppopt pragma -ppopt fill

Expert settings:

Command line pattern: ${COMMAND} -E ${FLAGS} ${INPUTS}

Table 3.172 lists and describes the preprocessor options for ARM.

Table 3.172 Tool Settings — ARM Preprocessor Options

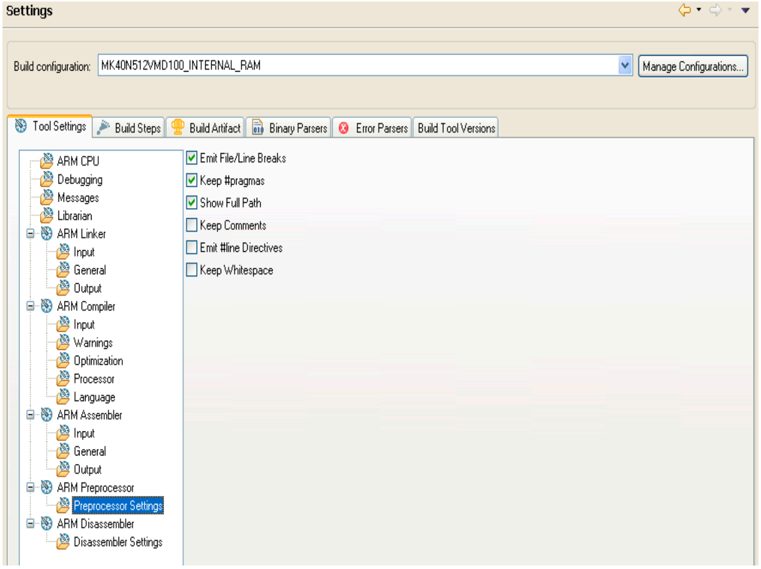
Option	Description
Command	Shows the location of the disassembler executable file
All options	Shows the actual command line the preprocessor will be called with
Expert Settings: Command line pattern	Shows the expert settings command line parameters; default `${COMMAND} -E${FLAGS} ${INPUTS}`

ARM Preprocessor > Preprocessor Settings

Use this panel to specify preprocessor behavior.

Figure 3.130 shows the **Preprocessor** panel.

Figure 3.186 Tool Settings — ARM Preprocessor> Preprocessor Settings


Settings
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Manage Configurations...
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Build configuration: MK40N512VMD100_INTERNAL_RAM

☑ Emit File/Line Breaks
☑ Keep #pragmas
☑ Show Full Path
☐ Keep Comments
☐ Emit #line Directives
☐ Keep Whitespace

Table 3.173 lists and describes the preprocessor options for ARM.

Table 3.173 Tool Settings — ARM Preprocessor Options

Option	Description
Emit file/line breaks	Check to notify file breaks (or #line breaks) appear in the output.
Keep #pragmas	Check to show pragma directives in the preprocessor output. Essential for producing reproducible test cases for bug reports.
Show full path	Check to display file changes in comments (as before) or in #line directives.
Keep comments	Check to display comments in the preprocessor output.

Table 3.173 Tool Settings — ARM Preprocessor Options (*continued*)

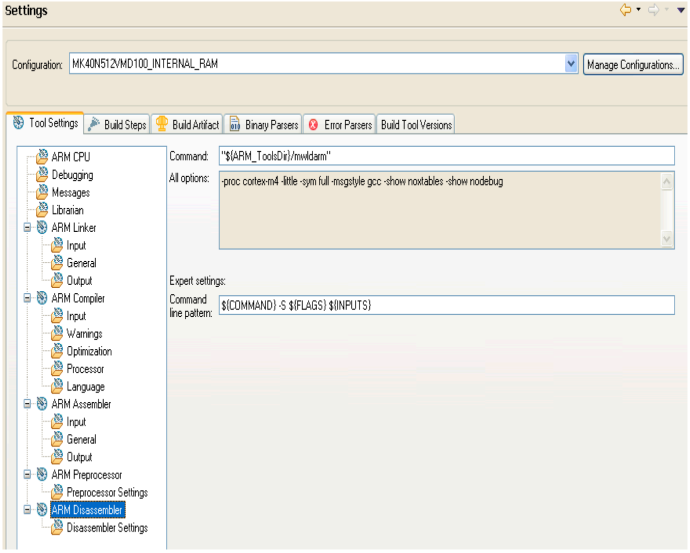
Option	Description
Emit #line directives	Check to display file changes in comments (as before) or in #line directives.
Keep whitespace	Check to copy whitespaces in preprocessor output. This is useful for keeping the starting column aligned with the original source, though the compiler attempts to preserve space within the line. This does not apply when macros are expanded.

ARM Disassembler

Use this panel to specify the command, options, and expert settings for **ARM Disassembler.**

Figure 3.187 shows the **ARM Disassembler** page.

Figure 3.187 Tool Settings — ARM Disassembler


Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Configuration: MK40N512VMD100_INTERNAL_RAM

Command: "${ARM_ToolsDir}/mwldarm"

All options: -proc cortex-m4 -little -sym full -msgstyle gcc -show noxtables -show nodebug

Expert settings:

Command line pattern: ${COMMAND} -S ${FLAGS} ${INPUTS}

Table 3.174 lists and describes the ARM disassembler options.

Table 3.174 Tool Settings — ARM Disassembler Options

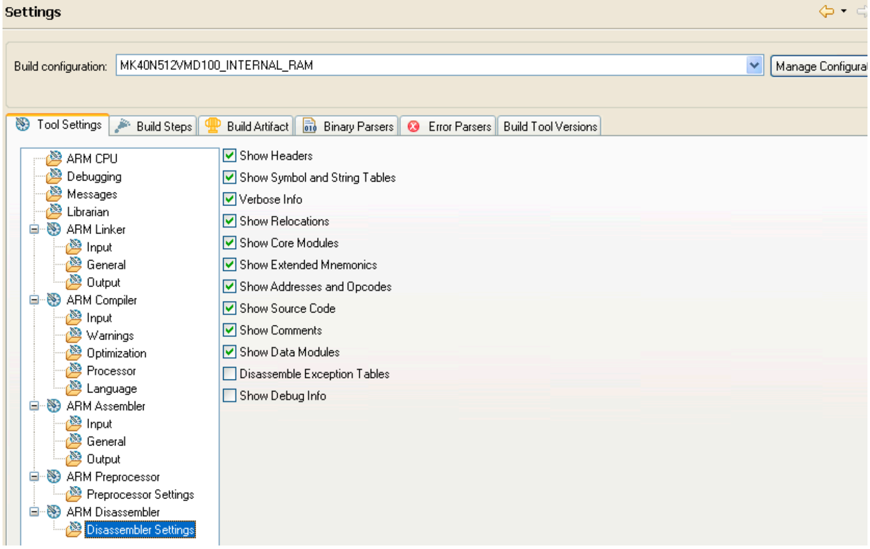
Option	Description
Command	Shows the location of the disassembler executable file
All options	Shows the actual command line the linker will be called with
Expert Settings Command line pattern	Shows the expert settings command line parameters; default is`${COMMAND} -dis ${FLAGS} ${INPUTS}`

ARM Disassembler > Disassembler Settings

Use this panel to control how the disassembler formats the listing file, as well as error and warning messages. You can specify verbosity of messages, whether to show headers, core modules, extended mnemonics, addresses, object or source code, ldata modules, exception tables, and debug information.

Figure 3.188 shows the **ARM Disassembler** settings.

Figure 3.188 Tool Settings — ARM Disassembler Settings


Settings
Manage Configura
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
ARM CPU
Debugging
Messages
Librarian
ARM Linker
Input
General
Output
ARM Compiler
Input
Warnings
Optimization
Processor
Language
ARM Assembler
Input
General
Output
ARM Preprocessor
Preprocessor Settings
ARM Disassembler
Disassembler Settings

Build configuration: MK40N512VMD100_INTERNAL_RAM

☑ Show Headers
☑ Show Symbol and String Tables
☑ Verbose Info
☑ Show Relocations
☑ Show Core Modules
☑ Show Extended Mnemonics
☑ Show Addresses and Opcodes
☑ Show Source Code
☑ Show Comments
☑ Show Data Modules
☐ Disassemble Exception Tables
☐ Show Debug Info

Table 3.175 lists and describes the ARM disassembler settings.

Table 3.175 Tool Settings — ARM Disassembler Options

Option	Description
Show Headers	Check to display headers in the listing file; disassembler writes listing headers, titles, and subtitles to the listing file
Show Symbol and String Tables	Check to display symbol and string tables directives to the listing file
Verbose Info	Check to shows each command line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emit
Show Relocations	Check to have the disassembler show information about relocated symbols. Clear to prevent the disassembler from showing information about relocated symbols.
Show Core Modules	Check to show core modules in the listing file
Show Extended Mnemonics	Check to show the extended mnemonics in the listing file
Show Addresses and Object Code	Check to show the addresses and object code in the listing file
Show Source Code	Check to show the source code in the listing file
Show Comments	Check to show the comments in the listing file
Show Data Modules	Check to show the data modules in the listing file
Disassemble Exception Tables	Check to disassemble exception tables in the listing file
Show Debug Info	Check to generate symbolic information for debugging the build target

4

Working with Debugger

A CodeWarrior project can have multiple associated launch configurations. A launch configuration is a named collection of settings that the CodeWarrior tools use.

The CodeWarrior project wizard generates launch configurations with names that follow the pattern ***projectname*** - ***configtype*** - ***targettype***, where:

- ***projectname*** represents the name of the project
- ***configtype*** represents the type of launch configuration
- ***targettype*** represents the type of target software or hardware on which the launch configuration acts

If you use the CodeWarrior wizard to create a new project, the IDE creates two debugger related launch configurations:

- ***Debug*** configuration — Produces unoptimized code for development purposes.
- ***Release*** configuration — Produces code intended for production purposes.

The topics in this chapter are:

- Initiating Debug Session
- Customizing Launch Configuration
- Debugging Bare Board Software
- Debugging Externally Built Executable Files

Initiating Debug Session

The CodeWarrior debugger provides three ways to initiate a debug session:

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download

NOTE These commands are available in the **Debug Configurations** dialog box. Select **Run > Debug Configurations** to open the **Debug Configurations** dialog box.

These commands differ in these ways:

- The **Attach** command establishes a debug session with code which is already running on the board. A target initialization file does not have to be run and the target is not reset, even if the launch configuration specifies this action. The debugger loads symbolic debugging information for the current build target's executable. No ELF file is downloaded and the program counter (PC) is not modified. The state of the running program is left undisturbed.

NOTE The debugger assumes that the current build target’s generated executable matches the code currently running on the target.

- The **Connect** command stops the target and (optionally) runs the target initialization file specified in the remote system configuration (RSE) to set up the board before connecting to it. The **Connect** function does not load any symbolic debugging information for the current build target’s executable. You therefore do not have access to source-level debugging and variable display. The **Connect** command resets the target if the launch configuration specifies this action. Further, the command stops the target, (optionally) runs an initialization script, does not load symbolics, download an ELF file, or modify the program counter (PC).
- The **Download** command stops the target, (optionally) runs the target initialization file, downloads the specified ELF file, and modifies the PC. The target is reset if the launch configuration specifies the action.

Table 4.1 Effect of Each Launch Configuration Type

Launch Configuration Type	Resets Target on Launch	Stops Target	Runs Init Script	Uses Symbolics	Modifies Entry PC	Downloads Application
Attach	Never	No	No	Yes	No	Never
Connect	Per Remote Connection setting: Usually set to Yes	Only if Reset on Launch	Per debugger Global Setting panel	No	Yes	Never
Download	Per Remote Connection setting: Usually set to Yes	Only if Reset on Launch	Per debugger Global Setting panel	Yes	Yes	Per HCS08/ RS08/ ColdFire Debugger Settings

Table 4.2 Connection Type Use Cases

Connection Type	Typical Use Example
Attach	Debug a target system without modifying its state at all initially, but allow use of symbolics during actual debug. Useful for debugging a system that is already up and running.
Connect	Raw debug of a board without any software or symbolics. Useful during hardware bring up, and often combined with scripts for checking various aspects of the hardware.
Download	Develop code that gets downloaded to the system on debugger launch. Useful for bare board code development without a working bootloader.

NOTE The default debugger configuration causes the debugger to cache symbolics between sessions. However, the **Connect** command invalidates this cache. If you must preserve the contents of the symbolics cache, and you plan to use the **Connect** command, uncheck the **Cache Symbolics Between Sessions** checkbox in the **Symbolics page** of the **Debug Configurations** dialog box just before you issue the **Connect** command.

CodeWarrior Attach

In a debugging session, the CodeWarrior Attach launch configuration skips setting up the target hardware, and downloading the program image to that target hardware. The code image might reside on the target hardware already, or you might want to skip setting up the target hardware. Like the CodeWarrior Connect launch configuration, the settings in the Arguments and Environment panels do not apply.

Although similar to a debugging session, the goal of attaching the debugger to a process is to get insight into the current state of that process, and to do so with minimal disturbance to its state of execution. Having the debugger attach to a process skips most of the state-altering steps involved in starting a debugging session, such as resetting the target, initializing the target, and downloading code. When the debugger finishes attaching to the process, you have many of the debugging capabilities that you would have in a debugging session (such as source-level debugging, line breakpoints, watchpoints, console input/output, and so on).

NOTE The debugger does not support restarting debugging sessions that you start by attaching the debugger to a process.

A process is an active program and related resources, such as:

- Executing program code
- Address space
- One or more threads of execution. A thread is a unit of activity that has a program counter and a set of processor registers
- Data section
- Set of resources, such as open files and pending signals

On a bare board (without an operating system), a given core has one process: one thread of execution executing one program in one address space. With an operating system, there can be several processes on a given core (with one active at any given moment).These processes either run different programs in different address spaces or even execute the same program, sharing an address space, open files, and so on.

You use the CodeWarrior **Debug Configurations** dialog box to view and attach to processes. To attach to a process:

1. In the **CodeWarrior Projects** view, select the project for which you want to start a debugging session in which you will attach to a process.
2. Select **Run > Debug Configurations**.

   The **Debug Configurations** dialog box appears.
3. Expand the **CodeWarrior Attach** group.
4. Click the **Main** tab.

NOTE If the **CodeWarrior Attach** group does not yet have any existing configuration, select **CodeWarrior Attach** and click the **New launch configuration** toolbar button of the **Debug Configurations** dialog box to create a new configuration.

5. Select an existing system within **Remote system** of which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*. Then specify parameters for the debugger and the target simulator or device with the **Launch Configuration Settings** by clicking the **Edit** button.
6. In the **Debugger** page, specify parameters for the debugger and the target simulator or device.
7. If you want to specify source lookup paths, click the **Source** tab and then use that page to specify the path information.
8. Click **Apply** to save your changes.
9. Click **Debug** to start the debugging session.

You just finished starting a debugging session and attaching the debugger to a process.

CodeWarrior Connect

In a debugging session, the **CodeWarrior Connect** launch configuration skips downloading the code image to the target hardware, and loading symbolics into the debugger. Skipping these steps is useful for board initialization and bringup. The code might reside on the target hardware already, or you might want to skip loading symbolics into the debugger.

Like the **CodeWarrior Attach** launch configuration, the settings in the **Arguments** and **Environment** panels do not apply. The **Source** tab is available, however, so that you can specify source paths in order to load an image after connecting the debugger to the target. Similar to starting a debugging session, you use the **Debug Configurations** dialog box to connect to a target:

1. In the C/C++ Projects view, select **Run > Debug Configurations**.
2. Expand the **CodeWarrior Connect** group.
3. Select an existing configuration from the expanded **CodeWarrior Connect** group.

NOTE If the **CodeWarrior Connect** group does not yet have any existing configuration, select **CodeWarrior Connect** and click the **New launch configuration toolbar** button of the **Debug Configurations** dialog box to create a new configuration.

4. Click the **Main** tab.
5. Select an existing system within **Remote system** of which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*. Then specify parameters for the debugger and the target simulator or device with the **Launch Configuration Settings** by clicking the **Edit** button.
6. If you want to specify source lookup paths, click the **Source** tab and then use that page to specify the path information.
7. Click **Debug** to start the debugging session.

You just finished starting a debugging session and connecting the debugger to the target.

CodeWarrior Download

In a debugging session, the **CodeWarrior Download** launch configuration downloads the code image to the target hardware, and loads symbolics into the debugger. The **Source** tab can be used to specify source paths in order to load an image after connecting the debugger to the target. Similar to starting a debugging session, you use the **Debug Configurations** dialog box to connect to a target:

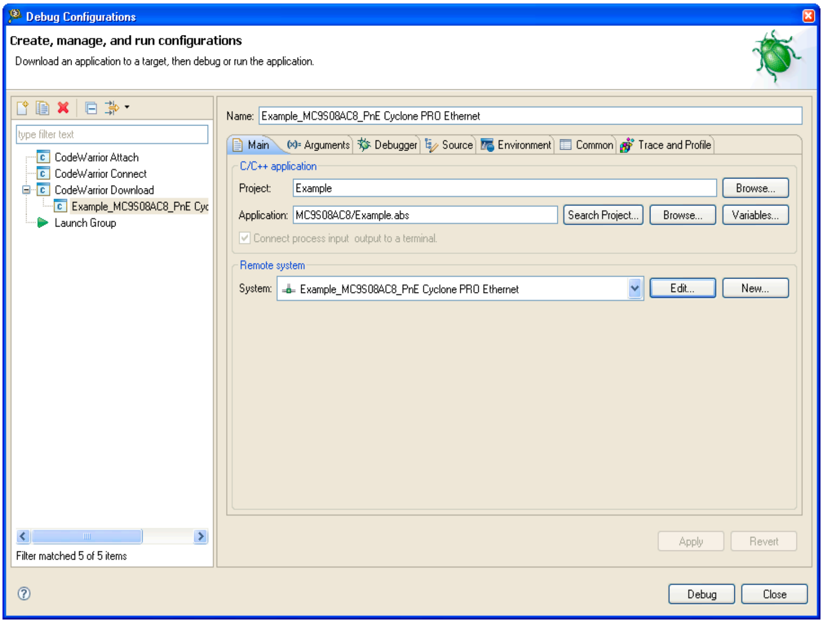
1. In the **C/C++ Projects** view, select **Run > Debug Configurations**.
2. Expand the **CodeWarrior Download** group.
3. Select an existing configuration from the expanded **CodeWarrior Download** group.

NOTE If the **CodeWarrior Download** group does not yet have any existing configuration, select the CodeWarrior Download title and click the **New launch configuration toolbar** button of the **Debug Configurations** dialog box to create a new configuration.

4. From the group of tabs in the upper-right side of the dialog box, click the **Debugger** tab.

   The debugger setting page appears in the area beneath the tabs. Figure 4.1 shows the **Debug Configurations** dialog box with CodeWarrior Download launch configurations.

Figure 4.1 CodeWarrior Download Launch Configurations



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - Example_MC9S08AC8_PnE Cyc
- Launch Group

Filter matched 5 of 5 items

Name: Example_MC9S08AC8_PnE Cyclone PRO Ethernet

C/C++ application

Project: Example

Application: MC9S08AC8/Example.abs

☑ Connect process input output to a terminal.

Remote system

System: Example_MC9S08AC8_PnE Cyclone PRO Ethernet

5. Change the settings on this page as per your requirements. For more information on customizing debugger settings, refer to the topic Customizing Launch Configuration.

6. If you want to specify source lookup paths, click the **Source** tab and then use that page to specify the path information.
7. Click **Apply** to save your changes.
8. Click **Debug** to start the debugging session.

You just finished starting a debugging session and connecting the debugger to the target.

Customizing Launch Configuration

The **Debug Configurations** dialog box contains seven tabs allowing you to customize all aspects of a launch configuration.

NOTE The CodeWarrior debugger shares some pages, such as **Connection** and **Download**. The settings that you specify in these pages also apply to the selected debugger.

The tabs in the **Debug Configurations** dialog box are:

- Main
- Arguments
- Debugger
- Source
- Environment
- Common
- Trace and Profile

NOTE As you modify a launch configuration's debugger settings, you create pending, or unsaved, changes to that launch configuration. To save the pending changes, you must click the **Apply** button of the **Debug Configurations** dialog box, or click the **Close** button and then the **Yes** button.

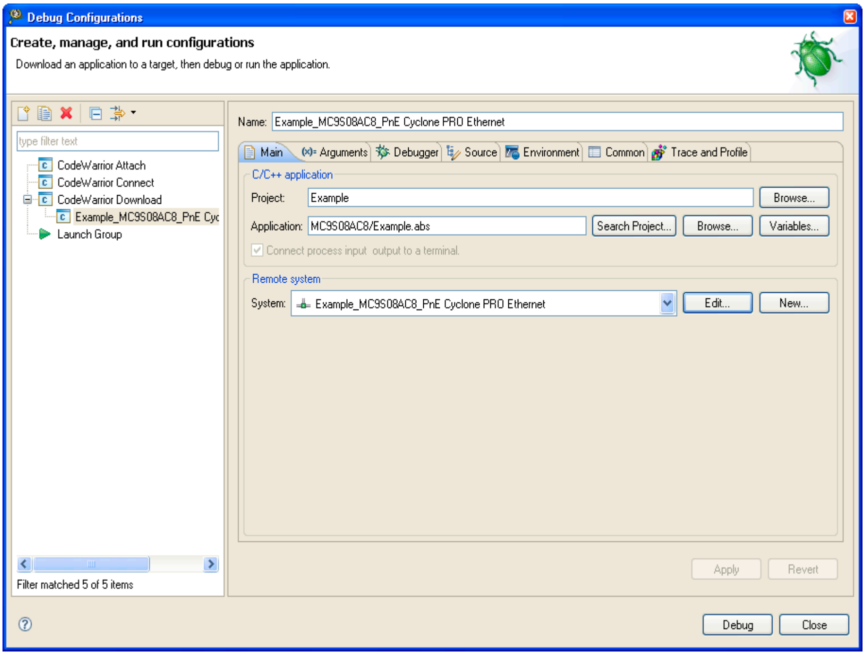
NOTE You can revert pending changes and restore their last saved settings. To undo pending changes, click the **Revert** button at the bottom of the **Debug Configurations** dialog box. The IDE restores the last set of saved settings to all pages of the **Debug Configurations** dialog box. Also, the IDE disables the **Revert** button until you make new pending changes.

Main

Use this page to specify the project and the application you want to run or debug.

The **Main** tab options are explained in Table 4.3.

Figure 4.2 Debug Configurations — Main Tab



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Example_MC9S08AC8_PnE Cyclone PRO Ethernet

C/C++ application

Project: Example

Application: MC9S08AC8/Example.abs

☑ Connect process input output to a terminal.

Remote system

System: Example_MC9S08AC8_PnE Cyclone PRO Ethernet

Filter matched 5 of 5 items

type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Example_MC9S08AC8_PnE Cyc
Launch Group
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
Browse...
Search Project...
Browse...
Variables...
Edit...
New...
Apply
Revert
Debug
Close

Table 4.3 Main Tab Options

Group	Option	Description
C/C++ application	Project	Specifies the name of the project.
	C/C++ Application	Specifies the name of the C or C++ application.
	Search Project	Click to open the **Program Selection** dialog box and select a binary.

Table 4.3 Main Tab Options (*continued*)

Group	Option	Description
Remote system	System	Specifies the applicable Remote System configuration.
	Edit	Click to edit the selected Remote System configuration. For more information on editing an existing remote system, refer to the topic Editing Remote System Connection.
	New	Click to create a new Remote System configuration for the selected project and application. For more information on creating a new remote system connection, refer to the topic Remote System Explorer in the *Freescale Eclipse Extensions Guide*.

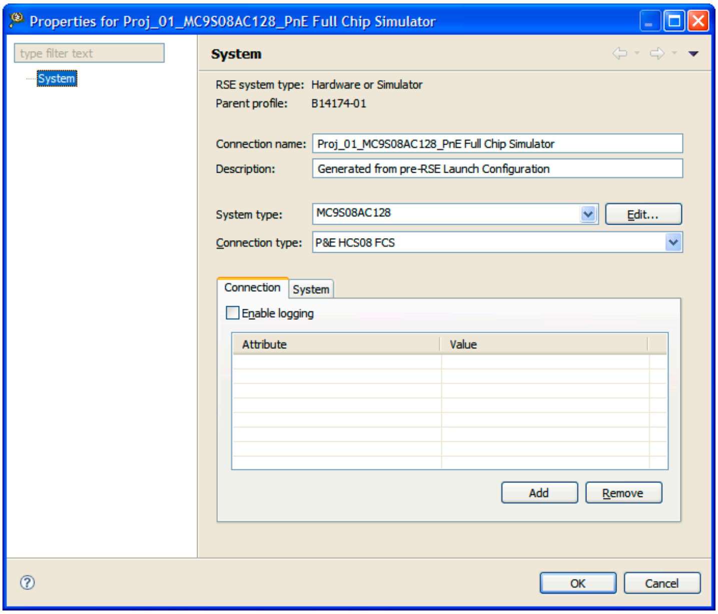
Editing Remote System Connection

To edit a remote system connection, click the **Edit** button in the **Main** tab of the **Debug Configurations** dialog box.

The **Properties for <*project_connection*>** dialog box appears. Here, **<*project_connection*>** is the name of the project followed by the associated connection. For example, *Proj_01_MC9S08AC128_PnE Full Chip Simulator*.

The remote system options in the **Properties for <*project_connection*>** dialog box change depending on the selected connection.

Figure 4.3 Properties for <project_connection> Dialog Box



Properties for Proj_01_MC9S08AC128_PnE Full Chip Simulator

RSE system type: Hardware or Simulator
Parent profile: B14174-01
Connection name: Proj_01_MC9S08AC128_PnE Full Chip Simulator
Description: Generated from pre-RSE Launch Configuration
System type: MC9S08AC128
Connection type: P&E HCS08 FCS

☐ Enable logging

Attribute	Value

[Table 4.4](#) lists the remote system options available in the **Properties for *<project_connection>*** dialog box.

Table 4.4 Properties for <project_connection> Dialog Box

Option	Description
RSE system type	Specifies whether the remote system explorer is a hardware or simulator.
Parent profile	Specifies the parent profile.
Connection name	Specifies the name of the connection used.
Description	Describes the connection used.
System type	Specifies CodeWarrior abstraction that represents the target processor layout. This can be a simple processor or a set of processors as defined by a configuration file.

Table 4.4 Properties for <project_connection> Dialog Box (*continued*)

Option	Description
Edit	Click to edit the system type. **Note:** For more information on editing system types, refer to the topic Editing System Types in the chapter Multicore Debugging.
Connection type	Specifies the type of the connection in use. The list is populated based on the selected remote system type and system type. Note: The options in the **Connection** sub tab will change depending on the selected connection.
Tabs	• **Connection** — Enables you to specify connection settings for the selected connection type. For more information refer to the topic Connection Tab Settings. • **Initialization** — Enables you to specify initialization settings for the selected connection type. For more information refer to the topic Initialization Tab Settings. • **System** — Enables you to specify system settings for the selected connection type. For more information refer to the topic System Tab Settings. • **Advanced** — Enables you to specify advanced settings for the selected connection type. For more information refer to the topic Advanced Tab Settings.
Buttons	• **OK** — Click to apply changes and close the **Properties for *<project_connection>*** dialog box. • **Cancel** — Click to close the **Properties for *<project_connection>*** dialog box without applying the changes.

Connection Tab Settings

Use this tab to specify the connection interface that the debugger uses to communicate with the connection on the target hardware. Table 4.5 lists the connection types available for the supported target boards. You can use the **Connections** tab to specify the connection interface that the debugger uses to communicate with the selected connection on the target hardware or target board.

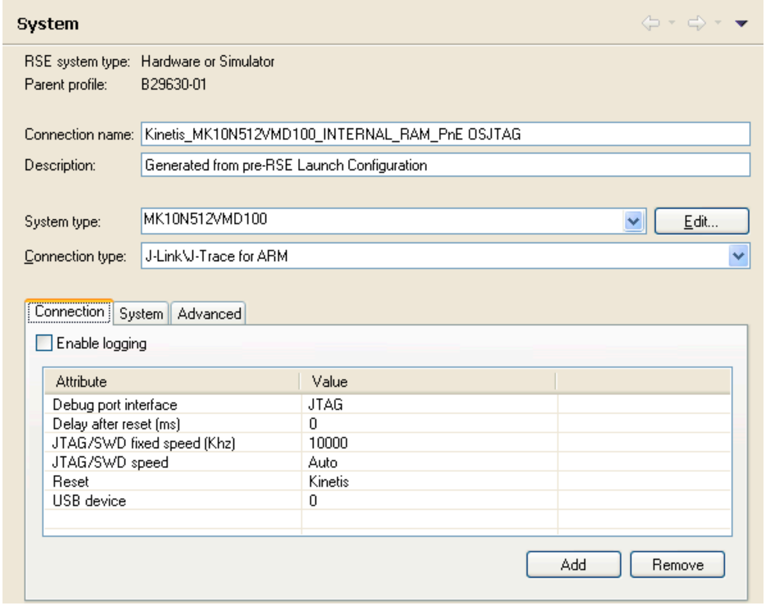
Table 4.5 Target Boards — Connection Types

Target Board	Connection Type
HCS08/RS08	• GDI Generic • OSBDM - JM60 for HCS08 • P&E HCS08 FCS • P&E HCS08 Multilink\Cyclone PRO • Softec
ColdFire V1\2\3\4	• Ethernet TAP • GDI Generic • OSBDM for ColdFire V1 • OSBDM - JM60 for ColdFire V2\3\4 • P&E ColdFire V1 Multilink\Cyclone PRO • P&E ColdFire V2\3\4 Multilink\Cyclone MAX • Softec • USB TAP
Kinetis	• J-Link\J-Trace for ARM • P&E ARM Multilink\Cyclone MAX
PowerPC OSJTAG	• GDI Generic • P&E PowerPCNexus Multilink/Cyclone MAX

J-Link\J-Trace for ARM

Use this setting if you manually launch and configure the connection-protocol service. It is used to communicate with the instruction set simulator and to configure hardware probes not pre-defined in the Physical Connection option. The **Connection** tab lists a table of named attributes and their assigned values. These attribute/value pairs are used to configure the characteristics of the interface.

Figure 4.4 J-Link\J-Trace for ARM — Connection Tab



RSE system type: Hardware or Simulator
Parent profile: B29630-01

Connection name: Kinetis_MK10N512VMD100_INTERNAL_RAM_PnE OSJTAG
Description: Generated from pre-RSE Launch Configuration

System type: MK10N512VMD100
Connection type: J-Link\J-Trace for ARM

☐ Enable logging

Attribute	Value
Debug port interface	JTAG
Delay after reset (ms)	0
JTAG/SWD fixed speed (Khz)	10000
JTAG/SWD speed	Auto
Reset	Kinetis
USB device	0

Table 4.7 lists the options available in the **Connection** tab.

Table 4.6 Connection Tab — J-Link\J-Trace for ARM

Option	Description
Enable Logging	Specifies if the debugger's communication is logged. Check this option to have the debugger's communications session logged to the **Console** in the **Debug** view. Clear this option to disable logging.
Attribute	Specifies an interface characteristic or option.
Value	Describes the current value or setting applied to the attribute.
Add	Click to add a new attribute/value pair. For more information, refer to the topic Add Attribute/Value.
Remove	Click to remove an existing attribute/value pair. For more information, refer to the topic Remove Attribute/Value.

The following topics explain how to add, modify, and delete entries in the Attribute/Value table:

- Add Attribute/Value
- Edit Existing Attribute/Value
- Remove Attribute/Value

Add Attribute/Value

To add a new attribute/value pair, perform these steps.

1. Click **Add**.

   The default entry `New attribute` appears in the **Attribute** column of the table, and the default entry `value` appears in the **Value** column. The new attribute's name is highlighted.

2. Specify a name for an attribute into this field, using alphanumeric characters only. The text replaces the default name.
3. Click the `<value>` field.

   The text in this text box is highlighted.

4. Specify a value to replace the default value.
5. To save the changes, click **Apply**.

Edit Existing Attribute/Value

To modify an attribute/value pair already in the table:

1. Click on an attribute name in the **Attribute** column of the table.

   The attribute name is highlighted.

2. Specify the name of the new attribute into this field, using alphanumeric characters only.

   The text replaces the previous name.

3. Click on the value field.

   The text in this text box is highlighted.

4. Specify a value to replace the previous value.
5. To save the changes, click **Apply**.

Remove Attribute/Value

To delete an existing attribute/pair in the table:

1. Double-click on the row of an attribute/pair in the table.

   The row is highlighted.

2. Click **Remove**.

   The row containing the selected attribute/value is deleted.

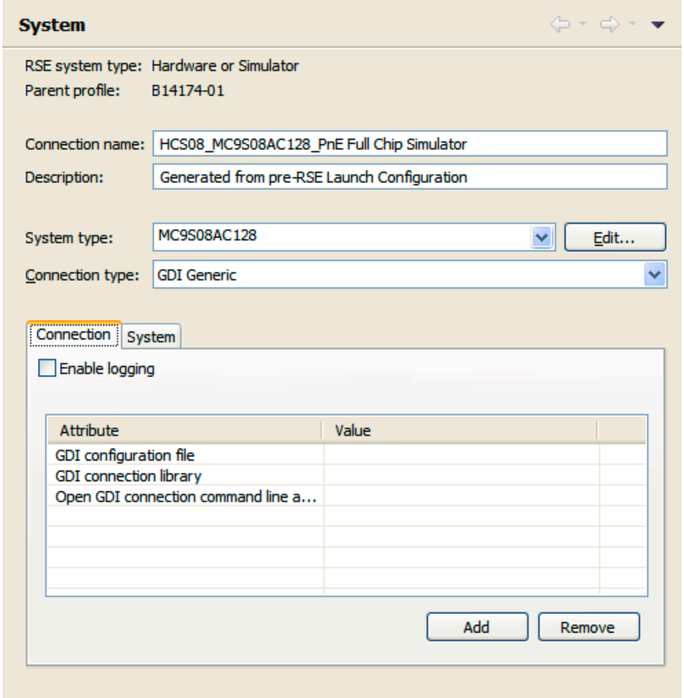
3. Click **Apply** to save any changes.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — Kinetis.

GDI Generic

Use this setting if you manually launch and configure the connection-protocol service (such as CCS), or if you are debugging code with a simulator. It is used to communicate with the instruction set simulator and to configure hardware probes not pre-defined in the Physical Connection option. The **Connection** tab lists a table of named attributes and their assigned values. These attribute/value pairs are used to configure the characteristics of the interface.

Figure 4.5 GDI Generic — Connection Tab



System

RSE system type: Hardware or Simulator
Parent profile: B14174-01

Connection name: HCS08_MC9S08AC128_PnE Full Chip Simulator
Description: Generated from pre-RSE Launch Configuration

System type: MC9S08AC128
Connection type: GDI Generic

☐ Enable logging

Attribute	Value
GDI configuration file	
GDI connection library	
Open GDI connection command line a...	

Table 4.7 lists the options available in the **Connection** tab.

Table 4.7 Connection Tab — GDI Generic

Option	Description
Enable Logging	Specifies if the debugger's communication is logged. Check this option to have the debugger's communications session logged to the **Console** in the **Debug** view. Clear this option to disable logging.
Attribute	Specifies an interface characteristic or option.
Value	Describes the current value or setting applied to the attribute.
Add	Click to add a new attribute/value pair. For more information, refer to the topic Add Attribute/Value.
Remove	Click to remove an existing attribute/value pair. For more information, refer to the topic Remove Attribute/Value.

The following topics explain how to add, modify, and delete entries in the Attribute/Value table:

- Add Attribute/Value
- Edit Existing Attribute/Value
- Remove Attribute/Value

Add Attribute/Value

To add a new attribute/value pair, perform these steps.

1. Click **Add**.

   The default entry `New attribute` appears in the **Attribute** column of the table, and the default entry `value` appears in the **Value** column. The new attribute's name is highlighted.

2. Specify a name for an attribute into this field, using alphanumeric characters only. The text replaces the default name.
3. Click the `<value>` field.

   The text in this text box is highlighted.

4. Specify a value to replace the default value.
5. To save the changes, click **Apply**.

Edit Existing Attribute/Value

To modify an attribute/value pair already in the table:

1. Click on an attribute name in the **Attribute** column of the table.

   The attribute name is highlighted.

2. Specify the name of the new attribute into this field, using alphanumeric characters only.

   The text replaces the previous name.

3. Click on the value field.

   The text in this text box is highlighted.

4. Specify a value to replace the previous value.
5. To save the changes, click **Apply**.

Remove Attribute/Value

To delete an existing attribute/pair in the table:

1. Double-click on the row of an attribute/pair in the table.

   The row is highlighted.

2. Click **Remove**.

   The row containing the selected attribute/value is deleted.

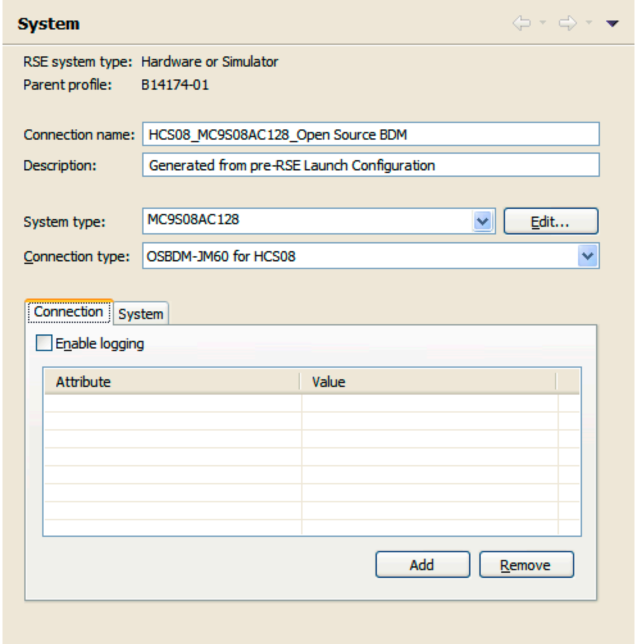
3. Click **Apply** to save any changes.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — HCS08 and Connections — RS08 .

OSBDM - JM60 for HCS08

The 8/16 bits debugger (and then the CodeWarrior IDE) can be connected to HCS08 hardware using the HCS08 OSBDM (Open Source BDM) cable. When the debugger runs the HCS08 Open Source BDM connection, it can communicate and debug HCS08 core-based hardware connected through the Open Source BDM Interface as described at the Freescale Semiconductor web site: `http://www.freescale.com` (keyword: OSBDM08).

Figure 4.6 OSBDM - JM60 for HCS08 — Connection Tab



RSE system type: Hardware or Simulator
Parent profile: B14174-01

Connection name: HCS08_MC9S08AC128_Open Source BDM
Description: Generated from pre-RSE Launch Configuration

System type: MC9S08AC128
Connection type: OSBDM-JM60 for HCS08

☐ Enable logging

Table 4.8 lists the options available in the **Connection** tab.

Table 4.8 Connection Tab — OSBDM - JM60 for HCS08

Option	Description
Enable Logging	Specifies if the debugger’s communication is logged. Check this option to have the debugger’s communications session logged to the **Console** in the **Debug** view. Clear this option to disable logging.
Attribute	Specifies an interface characteristic or option.
Value	Describes the current value or setting applied to the attribute.

Table 4.8 Connection Tab — OSBDM - JM60 for HCS08

Option	Description
Add	Click to add a new attribute/value pair. For more information, refer to the topic Add Attribute/Value.
Remove	Click to remove an existing attribute/value pair. For more information, refer to the topic Remove Attribute/Value.

The following topics explain how to add, modify, and delete entries in the Attribute/Value table:

- Add Attribute/Value
- Edit Existing Attribute/Value
- Remove Attribute/Value

Add Attribute/Value

To add a new attribute/value pair, perform these steps.

1. Click **Add**.

   The default entry `New attribute` appears in the **Attribute** column of the table, and the default entry `value` appears in the **Value** column. The new attribute's name is highlighted.

2. Specify a name for an attribute into this field, using alphanumeric characters only. The text replaces the default name.
3. Click the `<value>` field.

   The text in this text box is highlighted.

4. Specify a value to replace the default value.
5. To save the changes, click **Apply**.

Edit Existing Attribute/Value

To modify an attribute/value pair already in the table:

1. Click on an attribute name in the **Attribute** column of the table.

   The attribute name is highlighted.

2. Specify the name of the new attribute into this field, using alphanumeric characters only.

   The text replaces the previous name.

3. Click on the value field.

   The text in this text box is highlighted.

4. Specify a value to replace the previous value.
5. To save the changes, click **Apply**.

Remove Attribute/Value

To delete an existing attribute/pair in the table:

1. Double-click on the row of an attribute/pair in the table.

   The row is highlighted.

2. Click **Remove**.

   The row containing the selected attribute/value is deleted.

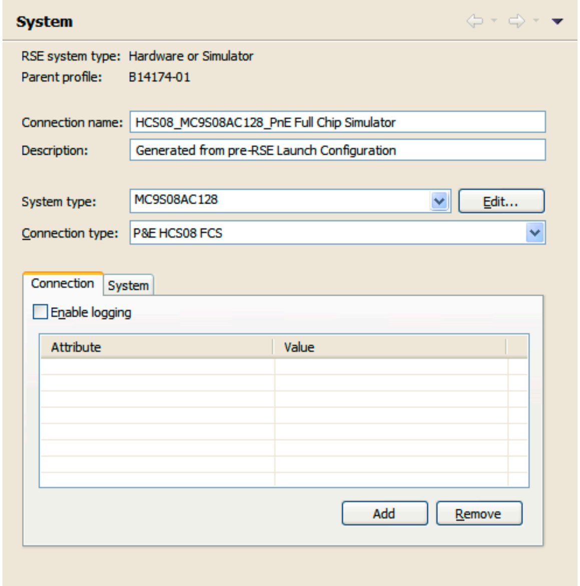
3. Click **Apply** to save any changes.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — HCS08 and Connections — RS08.

P&E HCS08 FCS

Use this setting to connect to P&E Full Chip Simulation with simulation of all chip peripherals. For more information refer to the topic P&E Full Chip Simulation.

Figure 4.7 P&E HCS08 FCS — Connection Tab



RSE system type: Hardware or Simulator
Parent profile: B14174-01
Connection name: HCS08_MC9S08AC128_PnE Full Chip Simulator
Description: Generated from pre-RSE Launch Configuration
System type: MC9S08AC128
Connection type: P&E HCS08 FCS
☐ Enable logging

Table 4.9 lists the options available in the **Connection** tab.

Table 4.9 Connection Tab — P&E HCS08 FCS

Option	Description
Enable Logging	Specifies if the debugger's communication is logged. Check this option to have the debugger's communications session logged to the **Console** in the **Debug** view. Clear this option to disable logging.
Attribute	Specifies an interface characteristic or option.
Value	Describes the current value or setting applied to the attribute.
Add	Click to add a new attribute/value pair. For more information, refer to the topic Add Attribute/Value.
Remove	Click to remove an existing attribute/value pair. For more information, refer to the topic Remove Attribute/Value.

The following topics explain how to add, modify, and delete entries in the Attribute/Value table:

- Add Attribute/Value
- Edit Existing Attribute/Value
- Remove Attribute/Value

Add Attribute/Value

To add a new attribute/value pair, perform these steps.

1. Click **Add**.

   The default entry `New attribute` appears in the **Attribute** column of the table, and the default entry `value` appears in the **Value** column. The new attribute's name is highlighted.

2. Specify a name for an attribute into this field, using alphanumeric characters only. The text replaces the default name.
3. Click the `<value>` field.

   The text in this text box is highlighted.

4. Specify a value to replace the default value.
5. To save the changes, click **Apply**.

Edit Existing Attribute/Value

To modify an attribute/value pair already in the table:

1. Click on an attribute name in the **Attribute** column of the table.

   The attribute name is highlighted.

2. Specify the name of the new attribute into this field, using alphanumeric characters only.

   The text replaces the previous name.

3. Click on the value field.

   The text in this text box is highlighted.

4. Specify a value to replace the previous value.
5. To save the changes, click **Apply**.

Remove Attribute/Value

To delete an existing attribute/pair in the table:

1. Double-click on the row of an attribute/pair in the table.

   The row is highlighted.

2. Click **Remove**.

   The row containing the selected attribute/value is deleted.

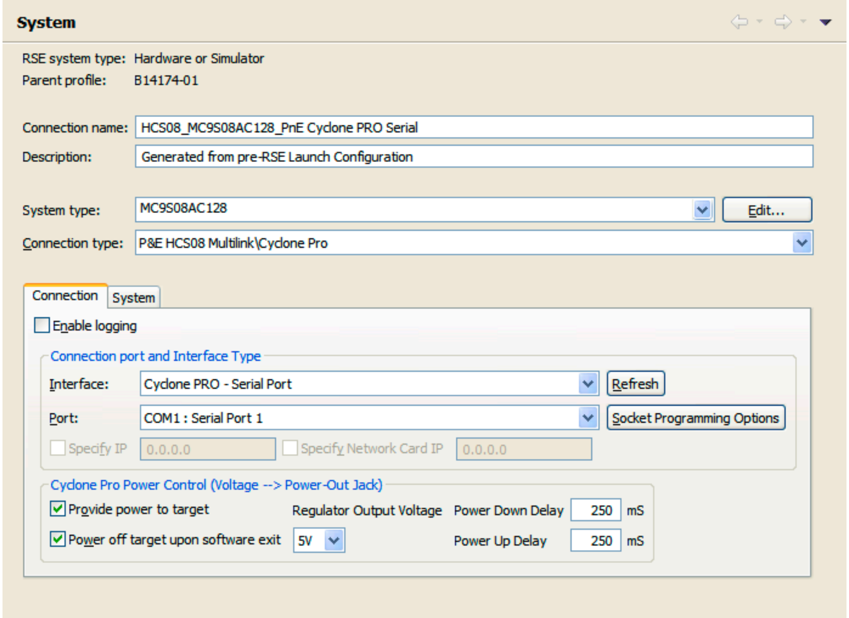
3. Click **Apply** to save any changes.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — HCS08 and Connections — RS08.

P&E HCS08 Multilink\Cyclone PRO

Use this setting to connect a P&E Multilink/Cyclone PRO probe with the target hardware.

Figure 4.8 P&E HCS08 Multilink\Cyclone PRO — Connection Tab



RSE system type: Hardware or Simulator
Parent profile: B14174-01
Connection name: HCS08_MC9S08AC128_PnE Cyclone PRO Serial
Description: Generated from pre-RSE Launch Configuration
System type: MC9S08AC128
Connection type: P&E HCS08 Multilink\Cyclone Pro

☐ Enable logging

Connection port and Interface Type
Interface: Cyclone PRO - Serial Port
Port: COM1 : Serial Port 1
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

For more information on HCS08, refer to the topics P&E HCS08 Multilink\Cyclone Pro\OSBDM and P&E HCS08 Multilink\Cyclone PRO/OSBDM Connection-Specific Options.

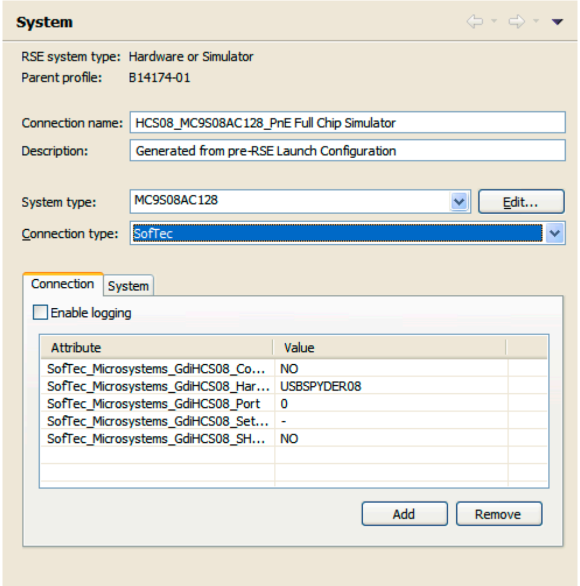
For information on RS08, refer to the topics P&E RS08 Multilink/Cyclone PRO/OSBDM and P&E RS08 Multilink/Cyclone PRO/OSBDM Connection-Specific Options.

Softec

Use this setting to connect a SofTec HCS08 probe with the target hardware. When the debugger runs the SofTec HCS08 connection, it can communicate and debug HCS08 core-based hardware connected through the SofTec in-circuit debugger/programmer units — SofTec Microsystems HCS08 ISP Debuggers/Programmers (inDART Series) and Starter Kits (PK and newer Series).

NOTE Refer to the inDART®-HCS08 In-Circuit Debugger/Programmer for Freescale HCS08 Family FLASH Devices User's Manual from SofTec for communication hardware requirements and SofTec product installation.

Figure 4.9 Softec — Connection Tab



RSE system type: Hardware or Simulator
Parent profile: B14174-01

Connection name: HCS08_MC9S08AC128_PnE Full Chip Simulator
Description: Generated from pre-RSE Launch Configuration

System type: MC9S08AC128
Connection type: SofTec

☐ Enable logging

Attribute	Value
SofTec_Microsystems_GdiHCS08_Co...	NO
SofTec_Microsystems_GdiHCS08_Har...	USBSPYDER08
SofTec_Microsystems_GdiHCS08_Port	0
SofTec_Microsystems_GdiHCS08_Set...	-
SofTec_Microsystems_GdiHCS08_SH...	NO

Table 4.10 lists the options available in the **Connection** tab.

Table 4.10 Connection Tab — Softec

Option	Description
Enable Logging	Specifies if the debugger’s communication is logged. Check this option to have the debugger’s communications session logged to the **Console** in the **Debug** view. Clear this option to disable logging.
Attribute	Specifies an interface characteristic or option.
Value	Describes the current value or setting applied to the attribute.
Add	Click to add a new attribute/value pair. For more information, refer to the topic Add Attribute/Value.
Remove	Click to remove an existing attribute/value pair. For more information, refer to the topic Remove Attribute/Value.

The following topics explain how to add, modify, and delete entries in the Attribute/Value table:

- Add Attribute/Value
- Edit Existing Attribute/Value
- Remove Attribute/Value

Add Attribute/Value

To add a new attribute/value pair, perform these steps.

1. Click **Add**.

   The default entry `New attribute` appears in the **Attribute** column of the table, and the default entry `value` appears in the **Value** column. The new attribute’s name is highlighted.

2. Specify a name for an attribute into this field, using alphanumeric characters only. The text replaces the default name.
3. Click the <`value`> field.

   The text in this text box is highlighted.

4. Specify a value to replace the default value.
5. To save the changes, click **Apply**.

Edit Existing Attribute/Value

To modify an attribute/value pair already in the table:

1. Click on an attribute name in the **Attribute** column of the table.

   The attribute name is highlighted.
2. Specify the name of the new attribute into this field, using alphanumeric characters only.

   The text replaces the previous name.
3. Click on the value field.

   The text in this text box is highlighted.
4. Specify a value to replace the previous value.
5. To save the changes, click **Apply**.

Remove Attribute/Value

To delete an existing attribute/pair in the table:

1. Double-click on the row of an attribute/pair in the table.

   The row is highlighted.
2. Click **Remove**.

   The row containing the selected attribute/value is deleted.
3. Click **Apply** to save any changes.

Ethernet TAP

The debugger uses an Ethernet TAP probe to connect to the target hardware. Enter in the Hostname/IP Address text box the host name or Internet Protocol address of the probe to which you want the debugger to connect By default, the Ethernet TAP probe uses Dynamic Host Configuration Protocol (DHCP) to acquire an IP address, netmask, and default gateway settings. If your network has a DHCP server which registers host names to the network’s name server, you can use the default host name to access the Ethernet TAP probe. Access is available when the HEARTBEAT LED starts flashing green.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter, Connections — ColdFire V1 and Connections — ColdFire V2/3/4.

OSBDM for ColdFire V1

This option specifies a hardware connection for OSBDM - ColdFire V1.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — ColdFire V1.

OSBDM - JM60 for ColdFire V2\3\4

This option specifies a hardware connection for OSBDM - JM60 ColdFire V2\3\4.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — ColdFire V2/3/4.

P&E ColdFire V1 Multilink\Cyclone PRO

This option specifies that the hardware connection is either a P&E Microsystems Multilink or a P&E Multisystems Cyclone PRO. The tab view displays the connection parameters this connection type uses.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — ColdFire V1.

P&E ColdFire V2\3\4 Multilink\Cyclone MAX

This option specifies that the hardware connection is either a P&E Microsystems Multilink or a P&E Multisystems Cyclone MAX. The tab view displays the connection parameters this connection type uses.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — ColdFire V2/3/4.

P&E ARM Multilink\Cyclone MAX

This option specifies that the hardware connection is either a P&E Microsystems Multilink or a P&E Multisystems Cyclone MAX. The tab view displays the connection parameters this connection type uses.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter Connections — Kinetis.

P&E PowerPCNexus Multilink/Cyclone MAX

This option specifies that the hardware connection is either a P&E Microsystems Multilink or a P&E Multisystems Cyclone MAX. The tab view displays the connection parameters this connection type uses.

NOTE For more information on MPC56xx, refer to the topics Connections — Power Architecture.

USB TAP

This option specifies that the physical connection uses USB TAP run control as the interface for debugging communications. The tab view displays the connection parameters this connection type uses.

NOTE For more information on the physical connection options presented with the option of probe, refer to the chapter, Connections — ColdFire V1 and Connections — ColdFire V2/3/4.

Initialization Tab Settings

Enables you to specify initialization settings for the selected connection type.

Table 4.11 Initialization Settings

Option	Description
Initialize Target	Specifies the target initialization file to be used by the debugger at the start of each debugging session. Check this option to activate the **Target Initialization File** text box where you can specify the path of the initialization file. Alternatively, you can specify the file path by using any of the buttons listed below: • **Workspace** — Opens a dialog box where you can specify the initialization file in terms of a location relative to the IDE's workspace directory. After you select the file, the path to that file appears in the **Target Initialization File** text box, relative to the path of the variable `workspace_loc`. The IDE resolves this variable to the absolute file system path of the workspace directory root. • **File System** — Opens a dialog box where you can browse for the initialization file. After you select the file, the absolute path to that file appears in the **Target Initialization File** text box. • **Variables** — Opens a dialog box where you can specify the initialization file in terms of IDE path variables. After you specify the file, the path to that file appears in the **Target Initialization File** text box, relative to the path variables that you use. The IDE resolves each path variable as explained in the **Variable Description** box at the bottom of the **Select Variable** dialog box. Clear this option if you want the debugger to use a default target initialization file.

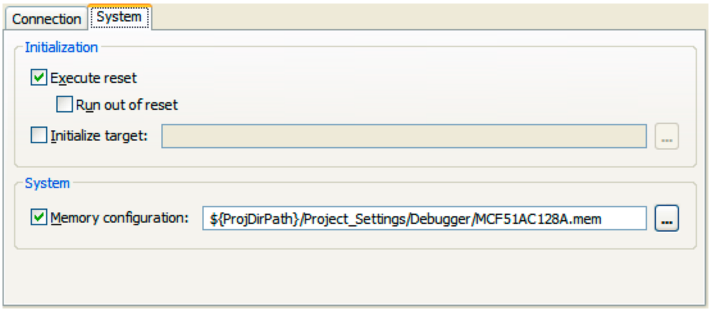
Table 4.11 Initialization Settings (*continued*)

Option	Description
Use Memory Configuration File	Specifies the memory configuration file to be used by the debugger at the start of each debugging session. Check this option to activate the **Memory Configuration File** text box where you can specify the path of the configuration file. Alternatively, you can specify the file path by using any of the buttons listed below: • **Workspace** — Opens a dialog box where you can specify the initialization file in terms of a location relative to the IDE's workspace directory. After you specify the file, the path to that file appears in the **Memory Configuration File** text box, relative to the path of the variable `workspace_loc`. The IDE resolves this variable to the absolute file system path of the workspace directory root. • **File System** — Opens a dialog box where you can browse for the initialization file. After you specify the file, the absolute path to that file appears in the **Memory Configuration File** text box. • **Variables** — Opens a dialog box where you can specify the initialization file in terms of IDE path variables. After you specify the file, the path to that file appears in the **Memory Configuration File** text box, relative to the path variables that you use. The IDE resolves each path variable as explained in the **Variable Description** box at the bottom of the **Select Variable** dialog box. Check the **Use Default** option to use the default memory configuration file and to deactivate the **Memory Configuration File** text box and the three buttons.

System Tab Settings

Enables you to specify system settings for the selected connection type.

Figure 4.10 System Tab Settings



☑ Execute reset

☐ Run out of reset

☐ Initialize target:

☑ Memory configuration: ${ProjDirPath}/Project_Settings/Debugger/MCF51AC128A.mem

Table 4.12 lists the available options in the **System** tab.

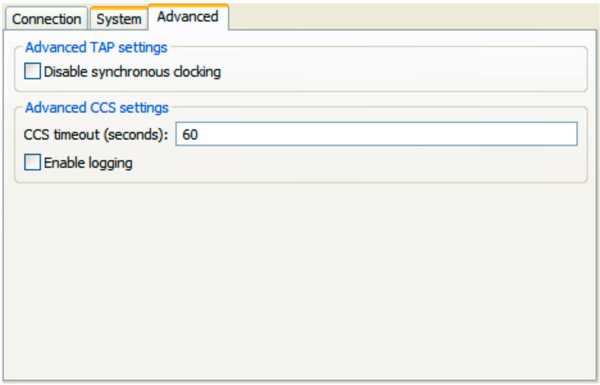
Table 4.12 System Settings

Option	Description
Initialization	
Execute Reset	Specifies that the debugger resets the target hardware before downloading a program for debugging purposes. Check this option to have the debugger reset the target before downloading the program to it. Clear this option to have the debugger download a program to the target without resetting that target.
Run out of reset	Determines what the ColdFire Microcontroller does after it is reset. Check this option to have the Microcontroller begin executing the program after it is reset. Clear this option to have the Microcontroller remain in debug mode after it is reset
Initialize target	Specifies the target initialization file to be used by the debugger at the start of each debugging session. Check this option to activate the adjacent text box where you can specify the path of the initialization file. Alternatively, you can click **...** (ellipsis) to open the **Initialize target** dialog box and specify the path of the file.
System	
Memory configuration	Specifies the memory configuration file to be used by the debugger at the start of each debugging session. Check this option to activate the adjacent text box where you can specify the path of the configuration file. Alternatively, you can click **...** (ellipsis) to open the **Memory configuration** dialog box and specify the path of the file.

Advanced Tab Settings

Enables you to specify advanced settings for the selected connection type.

Figure 4.11 Advanced Tab Settings



Advanced TAP settings

☐ Disable synchronous clocking

Advanced CCS settings

CCS timeout (seconds): 60

☐ Enable logging

Table 4.12 lists the available options in the **Advanced** tab.

Table 4.13 Advanced Settings

Option	Description
Advanced TAP settings	
Disable synchronous clocking	Check this option to have the debugger use a standard (slow) procedure to write to memory on the target system. Clear this option to have the debugger use an optimized (fast) download procedure to write to memory on the target system. The fast download mechanism is used by default when writing to target memory. Check this option if the fast download procedure results in load failures.
Advanced CCS settings	

Table 4.13 Advanced Settings (*continued*)

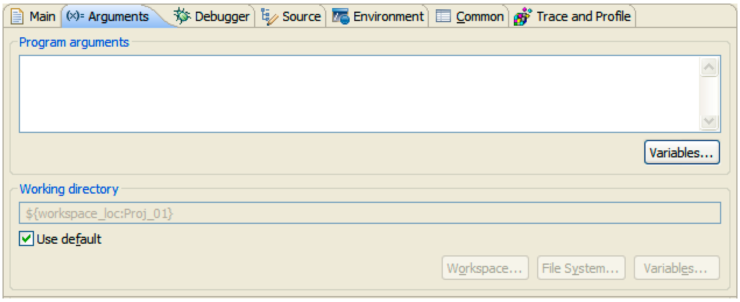
Option	Description
CCS timeout (seconds)	Enter the number of seconds after which you want the debugger to treat CCS as unresponsive. The debugger treats the time interval that you specify as a window of validity in which CCS must complete debugger requests. If CCS does not complete the requests during the specified time interval, the debugger treats CCS as unresponsive. For example, you might specify 30 seconds to give intensive CCS operations enough time to succeed during a debugging session. You do not want to wait for 30 seconds for the initial connect operation, if you mistyped the Ethernet TAP probe's IP address, or forgot to turn on the target hardware. For these reasons, the debugger treats the specified value differently for initial-connect operations.
Enable logging	Check this option to have the debugger output connection protocol activity to a console in the Console view. Clear this option if you do not want the debugger to output connection-protocol activity to a console.

Arguments

Use this page to specify the program arguments that an application uses and the working directory for a run or debug configuration.

The **Arguments** tab options are explained in Table 4.14.

Figure 4.12 Debug Configurations — Arguments Tab



Program arguments

Working directory

${workspace_loc:Proj_01}

☑ Use default

Table 4.14 Arguments Tab Options

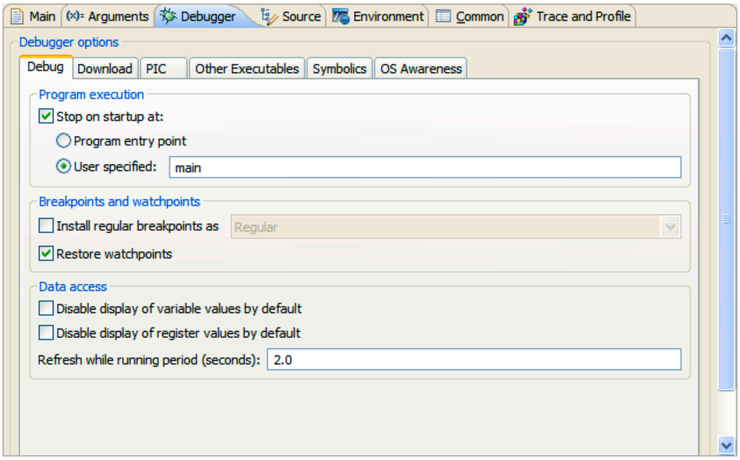
Option	Description
Program arguments	Specifies the arguments passed on the command line.
Variables	Click to select variables by name to include in the program arguments list.
Working Directory	Specifies the run/debug configuration working directory.
Use default	Check to specify the local directory or uncheck to specify a different workspace, a file system location, or variable.
Workspace	Click to specify the path of, or browse to, a workspace relative working directory.
File System	Click to specify the path of, or browse to, a file system directory.
Variables	Click to specify variables by name to include in the working directory.

Debugger

Use this page to select a debugger to use when debugging an application. The **Debugger** tab presents different sub-tabs for specifying different settings.

NOTE The sub-tabs under the **Debugger** tab change depending on the derivative and connection you select while creating the project.

Figure 4.13 Debug Configurations — Debugger Tab


Main
(x)= Arguments
Debugger
Source
Environment
Common
Trace and Profile
Debug
Download
PIC
Other Executables
Symbolics
OS Awareness

Debugger options

Program execution

☑ Stop on startup at:

○ Program entry point

● User specified: main

Breakpoints and watchpoints

☐ Install regular breakpoints as Regular

☑ Restore watchpoints

Data access

☐ Disable display of variable values by default

☐ Disable display of register values by default

Refresh while running period (seconds): 2.0

NOTE The subsequent topics explain the various settings that you can configure using the pages beneath the **Debugger** tab.

The sub-tabs available under the **Debugger** tab are:

- Debug
- Download
- PIC
- Other Executables
- Symbolics
- OS Awareness
- Exceptions
- Reset
- Interrupts
- EPPC Exceptions
- System Call Services

Debug

Use this page to specify the program execution options, breakpoint, watchpoint options, and target access behavior.

The **Debug** tab options are explained in Table 4.15.

Table 4.15 Debug Tab Options

Group	Option	Description
Program execution	Stop on startup at	Stops program at specified location. when unchecked, the program runs until you interrupt it manually, or until it hits a breakpoint.
	Program entry point	Select to stop the debugger at a specified program entry point. Click **Advanced** to modify the default program entry points.
	User specified	Select to stop the debugger at a user-specified function. The default location is `main`.
Breakpoints and watchpoints	Install regular breakpoints as	Check this option to install breakpoints as either: • Regular, or • Hardware, or • Software Clear this option to install breakpoints as Regular breakpoints.
	Restore watchpoints	Check this option to restore previous watchpoints.
Data access	Disable display of variable values by default	Check this option to disable the display of variable values. Clear this option to enable the display of variable values.
	Disable display of register values by default	Check this option to disable the display of register values. Clear this option to enable the display of register values.
	Refresh while running period (seconds)	Specifies the referesh period used when a view is configured to refresh while the application is running.

Download

Use this tab to specify the program sections the debugger downloads to the target, and whether the debugger should read back those sections and verify them.

NOTE Checking all checkboxes in the **Program Download Options** group significantly increases download time.

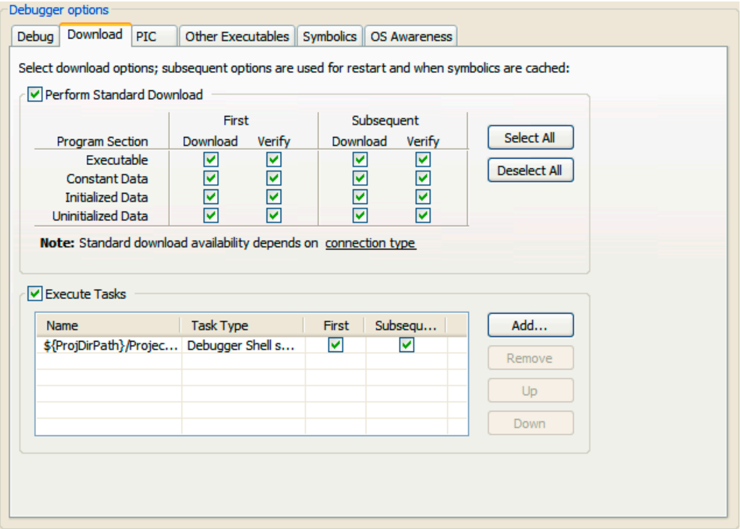
Figure 4.14 shows the Download tab. Briefly, the section data types are:

- **Executable** — These sections contain your program's code.
- **Constant Data** — These sections contain your program's constants. These values can not be modified.
- **Initialized Data** — Initialized data sections contain your program's modifiable data.
- **Uninitialized Data** — Uninitialized data sections contain your program's uninitialized variables.

Table 4.16 explains each option.

First options apply to the first debugging session. Subsequent options apply to successive debugging sessions. The **Download** options control whether the debugger downloads the specified section data type to the target hardware. The **Verify** options control whether the debugger reads the specified section data type from the target hardware and compares the data read against the data written to the device.

Figure 4.14 Download Tab



Select download options; subsequent options are used for restart and when symbolics are cached:

☑ Perform Standard Download

	First		Subsequent	
Program Section	Download	Verify	Download	Verify
Executable	☑	☑	☑	☑
Constant Data	☑	☑	☑	☑
Initialized Data	☑	☑	☑	☑
Uninitialized Data	☑	☑	☑	☑

Note: Standard download availability depends on connection type

☑ Execute Tasks

Name	Task Type	First	Subsequ...
${ProjDirPath}/Projec...	Debugger Shell s...	☑	☑

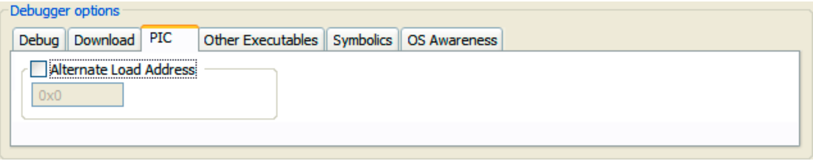
Table 4.16 Download Tab Settings

Section Data Type	Description
Executable	Controls downloading and verification for executable sections. Check appropriate checkboxes to specify downloading and verifications, for initial launch and for successive runs.
Constant Data	Controls downloading and verification for constant-data sections. Check appropriate checkboxes to specify downloading and verifications, for initial launch and for successive runs.
Initialized Data	Controls downloading and verification for initialized-data sections. Check appropriate checkboxes to specify downloading and verifications, for initial launch and for successive runs.
Uninitialized Data	Controls downloading and verification for uninitialized-data sections. Check appropriate checkboxes to specify downloading and verifications, for initial launch and for successive runs.
Select All	Selects all of the options available for downloading and verifying the program.
Deselect All	The debugger will not download or verify any program sections.

PIC

Use this tab to specify an alternate address for the debugger to load a Position Independent Code (PIC) module on a target board. at an different address than specified in the ELF file. Also, when having to debug an application (such as U-Boot) built with ROM addresses after it has relocated itself to RAM. Usually, PIC is linked in such a way that the entire image starts at address 0x00000000. The **PIC** tab lets you specify an alternate address at which the debugger will load the PIC module in target memory. Figure 4.15 shows the **PIC** tab.

Figure 4.15 PIC Tab


Debugger options
Debug
Download
PIC
Other Executables
Symbolics
OS Awareness

☐ Alternate Load Address

0x0

Check the **Alternate Load Address** option and then enter the address (in hexadecimal notation) in the corresponding text box. The address that you specify is the starting address at which the debugger loads your program or finds it after runtime relocation. Specifying an alternate load address lets the debugger map the symbolic debugging information contained in the original ELF file to the relocated application image in RAM.

NOTE The debugger does not verify whether your code can execute at the specified address. As a result, the PIC generation settings of the compiler, linker and your program's startup routines must correctly set any base registers and perform any required relocations.

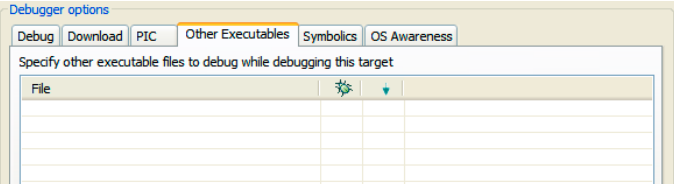
Clear the **Alternate Load Address** option to have the debugger load your program at a default starting address.

Other Executables

Use this tab to specify additional ELF files to download or debug in addition to the main executable file associated with the launch configuration.

Figure 4.16 shows the **Other Executables** tab view.

Figure 4.16 Other Executables Tab


Debugger options
Debug
Download
PIC
Other Executables
Symbolics
OS Awareness

Specify other executable files to debug while debugging this target

File		

Table 4.17 describes the **Other Executables** debugger settings.

Table 4.17 Other Executables Tab Settings

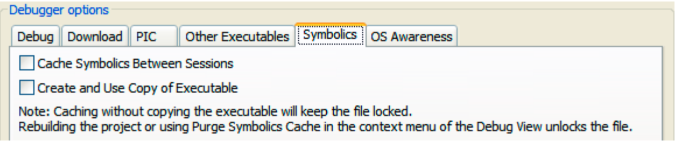
Option	Description
File list	Shows files and projects that the debugger uses during each debug session The **Debug** column (🐞) — If this option is checked the debugger loads symbolics for the file. If you clear this option, the IDE does not load symbolics for the file. The **Download** column (↓) — If this option is checked the debugger downloads the file to the target device. If you clear this option, the debugger does not download the file to the target device.
Add	Click to open the **Debug Other Executable** dialog box. Use the dialog box to specify the following settings: • Specify the location of the additional executable — Enter the path to the executable file that the debugger controls in addition to the current project's executable file. Alternatively, click **Browse** to specify the file path. • Load symbols — Check this option to have the debugger load symbols for the specified file. Clear to prevent the debugger from loading the symbols. The **Debug** column of the **File** list corresponds to this setting. • Download to device — Check this option to have the debugger download the specified file to the target device. Clear this option to prevent the debugger from downloading the file to the device. The **Download** column of the **File** list corresponds to this setting. • OK — Click to add the information that you specify in the **Debug Other Executable** dialog box to the **File** list.
Change	Click to open the **Debug Other Executable** dialog box. The dialog box shows the current settings for the selected executable file in the **File** list column. Change this information as required and click **OK** to update the entry in the **File** list.
Remove	Click to remove the entry currently selected in the File list.

Symbolics

Use this tab to specify whether the debugger keeps symbolics in memory. Symbolics represent an application's debugging and symbolic information. Keeping symbolics in memory, known as caching symbolics, is beneficial when you debug a large-size application. Figure 4.17 shows the **Symbolics** tab.

Consider a situation in which the debugger loads symbolics for a large application, but does not download content to a hardware device and the project uses custom makefiles with several build steps to generate this application. In such a situation, caching symbolics helps speed up the debugging process. The debugger uses the readily available cached symbolics during subsequent debugging sessions. Otherwise, the debugger spends significant time creating an in-memory representation of symbolics during subsequent debugging sessions.

Figure 4.17 Symbolics Tab



☐ Cache Symbolics Between Sessions

☐ Create and Use Copy of Executable

Note: Caching without copying the executable will keep the file locked.
Rebuilding the project or using Purge Symbolics Cache in the context menu of the Debug View unlocks the file.

Table 4.18 describes the **Symbolics** debugger settings.

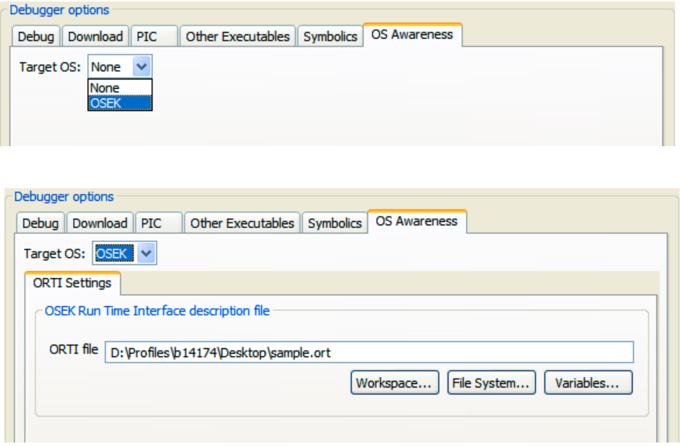
Table 4.18 Symbolics Tab Settings

Option	Description
Cache Symbolics Between Sessions	Check this option to have the debugger cache symbolics between debugging sessions. With Create and Use Copy of Executable cleared, the executable file remains locked after the debugging session ends. In the **Debug** view, right-click the locked file and select **Un-target Executables** to have the debugger delete its symbolics cache and release the file lock. The IDE enables this menu command when there are currently unused cached symbolics that it can purge. Clear this option so that the debugger does not cache symbolics between debugging sessions.
Create and Use Copy of Executable	Check this option to have the debugger create and use a copy of the executable file. Using the copy helps avoid file-locking issues with the build system. If you check this checkbox, the IDE can build the executable file in the background during a debugging session. Clear this option so that the debugger does not create and use a copy of the executable file.

OS Awareness

Use this tab to specify the operating system (OS) that resides on the target device. Figure 4.18 shows the **OS Awareness** tab view.

Figure 4.18 OS Awareness Tab — ColdFire


Debugger options
Debug
Download
PIC
Other Executables
Symbolics
OS Awareness
None
OSEK
ORTI Settings
OSEK Run Time Interface description file
Workspace...
File System...
Variables...

Target OS: None

Target OS: OSEK

ORTI file D:\Profiles\b14174\Desktop\sample.ort

Use the **Target OS** list box to specify the OS that runs on the target device, or specify None to have the debugger use the bare board.

If you select OSEK as the target OS, you are required to specify the path of the OSEK Run Time Interface (ORTI) description file. You can click any of the following to navigate and browse to the ORTI file.

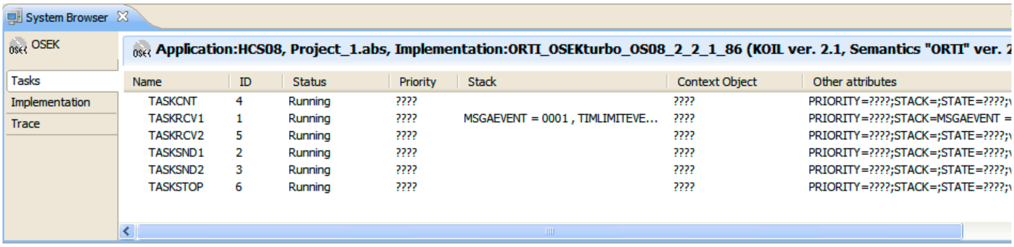
- Workspace — Click to open the **Folder Selection** dialog box and select a workspace location for the project. This is the directory that will contain the plug-ins and features to build, including any generated artifacts.
- File system — Click to open the **Browse For Folder** dialog box and select a folder.
- Variables — Click to open the **Select build variable** dialog box and select a variable to specify as an argument for the build directory, or create and configure simple build variables which you can reference in build configurations that support variables.

Selecting OSEK visualizes the RTOS's internal objects, like tasks, alarms, counters, and resources. Other objectives are:

- contains basic trace implementation that records and displays the last N values of the traceable attributes.
- enables multiple debuggers and handle multiple OSEK applications.
- enables components reuse.

Once the ORTI file is specified, click **Apply** and **Debug**. The ORTI file is automatically loaded at process creation time. Check the OSEK console to ensure whether the file was loaded or not. Any I/O exceptions or parse errors will be spilled out in the console. Once the ORTI file is successfully loaded, the **System Browser** view appears as in Figure 4.19

Figure 4.19 System Browser View



Application:HCS08, Project_1.abs, Implementation:ORTI_OSEKturbo_OS08_2_2_1_86 (KOIL ver. 2.1, Semantics "ORTI" ver.

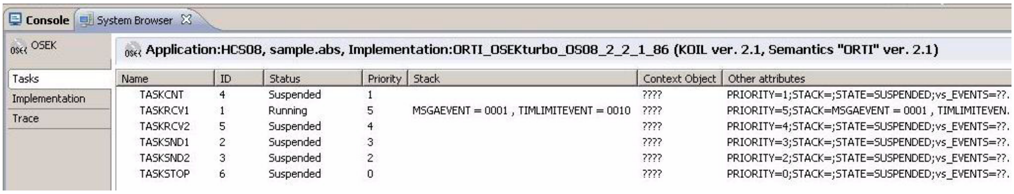
Name	ID	Status	Priority	Stack	Context Object	Other attributes
TASKCNT	4	Running	????		????	PRIORITY=????;STACK=;STATE=????;
TASKRCV1	1	Running	????	MSGAEVENT = 0001 , TIMLIMITEVE...	????	PRIORITY=????;STACK=MSGAEVENT =
TASKRCV2	5	Running	????		????	PRIORITY=????;STACK=;STATE=????;
TASKSND1	2	Running	????		????	PRIORITY=????;STACK=;STATE=????;
TASKSND2	3	Running	????		????	PRIORITY=????;STACK=;STATE=????;
TASKSTOP	6	Running	????		????	PRIORITY=????;STACK=;STATE=????;

The three tabs in the **System Browser** view are: Tasks, Implementation, and Trace.

Tasks

The **Tasks** tab is a built-in view tab which lists only basic OS info like: task name, id, priority and state. For full information on application task open the **Implementation** tab.

Figure 4.20 System Browser View — Tasks



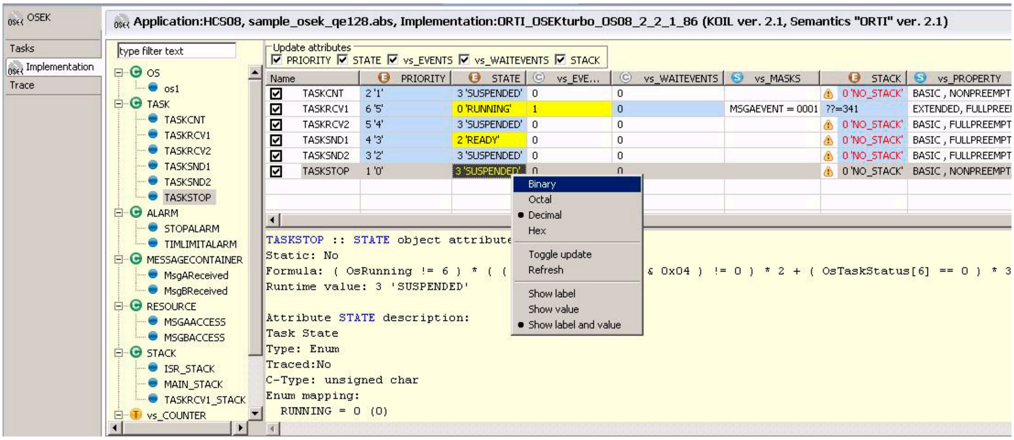
Application:HCS08, sample.abs, Implementation:ORTI_OSEKturbo_OS08_2_2_1_86 (KOIL ver. 2.1, Semantics "ORTI" ver. 2.1)

Name	ID	Status	Priority	Stack	Context Object	Other attributes
TASKCNT	4	Suspended	1		????	PRIORITY=1;STACK=;STATE=SUSPENDED;vs_EVENTS=??.
TASKRCV1	1	Running	5	MSGAEVENT = 0001 , TIMLIMITEVENT = 0010	????	PRIORITY=5;STACK=MSGAEVENT = 0001 , TIMLIMITEVEN.
TASKRCV2	5	Suspended	4		????	PRIORITY=4;STACK=;STATE=SUSPENDED;vs_EVENTS=??.
TASKSND1	2	Suspended	3		????	PRIORITY=3;STACK=;STATE=SUSPENDED;vs_EVENTS=??.
TASKSND2	3	Suspended	2		????	PRIORITY=2;STACK=;STATE=SUSPENDED;vs_EVENTS=??.
TASKSTOP	6	Suspended	0		????	PRIORITY=0;STACK=;STATE=SUSPENDED;vs_EVENTS=??.

Implementation

The **Implementation** tab lists the full kernel object structure of the OSEK application and is composed of the Kernel objects tree and Kernel type viewer panels, refer topics Kernel Objects Tree Panel and Kernel Type Viewer Panel.

Figure 4.21 System Browser View — Implementation


OSEK
Tasks
Implementation
Trace
type filter text
OS
os1
TASK
TASKCNT
TASKRCV1
TASKRCV2
TASKSND1
TASKSND2
TASKSTOP
ALARM
STOPALARM
TIMLIMITALARM
MESSAGECONTAINER
MsgAReceived
MsgBReceived
RESOURCE
MSGAACCESS
MSGBACCESS
STACK
ISR_STACK
MAIN_STACK
TASKRCV1_STACK
vs_COUNTER
Binary
Octal
● Decimal
Hex
Toggle update
Refresh
Show label
Show value
● Show label and value

Application:HCS08, sample_osek_qe128.abs, Implementation:ORTI_OSEKturbo_OS08_2_2_1_86 (KOIL ver. 2.1, Semantics "ORTI" ver. 2.1)

Update attributes
☑ PRIORITY ☑ STATE ☑ vs_EVENTS ☑ vs_WAITEVENTS ☑ STACK

	Name	PRIORITY	STATE	vs_EVE...	vs_WAITEVENTS	vs_MASKS	STACK	vs_PROPERTY
☑	TASKCNT	2 '1'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , NONPREEMPT
☑	TASKRCV1	6 '5'	0 'RUNNING'	1	0	MSGAEVENT = 0001	??=341	EXTENDED, FULLPREEI
☑	TASKRCV2	5 '4'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSND1	4 '3'	2 'READY'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSND2	3 '2'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSTOP	1 '0'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , NONPREEMPT

```
TASKSTOP :: STATE object attribute
Static: No
Formula: ( OsRunning != 6 ) * ( (                          & 0x04 ) != 0 ) * 2 + ( OsTaskStatus[6] == 0 ) * 3
Runtime value: 3 'SUSPENDED'

Attribute STATE description:
Task State
Type: Enum
Traced:No
C-Type: unsigned char
Enum mapping:
  RUNNING = 0 (0)
```

Kernel Objects Tree Panel

Lists all kernel objects and their kernel types. To differentiate standard and vendor defined types a different icon is used. Each selection is reflected in the Kernel type viewer.

Kernel Type Viewer Panel

Lists all kernel objects of a certain kernel type. Each kernel object is described by its attributes.

- Update attributes panel — The role of this panel is to update kernel type attribute on each kernel object based on the next controller request. In UI terms, if an attribute is cleared the entire column is not renewed on the next update request. If the kernel type attribute is a String or the entire column contains only static attributes then the implementation attribute does not appear in this panel. Similarly, each table item have a checkbox which will prohibit the attribute update only for this kernel object.
- ORTI item description viewer. This is the lower text pane which displays static (or information extracted about that item from the ORTI file) and dynamic (runtime values) data. The ORTI description viewer also lists tags or links. A tag or link can be any ORTI item, that is any kernel object, type or attribute name. At load time these names are put in global cache along with their ORTI entity. They are marked as hyperlinks and the open action will select that kernel type or object.
- Kernel objects detailed viewer — This table hold instances of the same kernel type. Intuitively, each table cell represents a kernel object attribute; except the first column which is allocated for object name. The table cells hold some graphical properties.

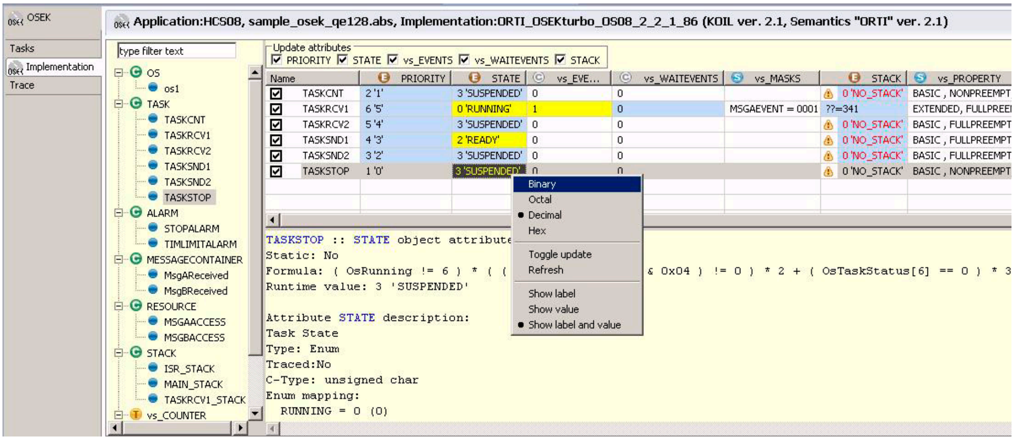
Table 4.19 Graphical Properties of Kernel Objects Detailed Viewer

Option	Description
Backgroung Color	
White	Indicates static attributes like strings or constant C type or enum values; default.
Blue	Indicates dynamic attributes that need to be computed through C formulas or expressions; they are marked this way in order to distinguish them from static attributes.
Yellow	Indicates dynamic attributes to denote a value change.
Black	Indicates the table cursor or which cell is currently selected.
Foreground color	
Black	Indicates default text color
Red	Indicates an error or a value inconsistency.
Icons	Signifies the attribute type (Enum, String or C type) and the cells can a warning icon which indicates a possible error or an invalid value; table columns are appended with an image. For attribute SERVICETRACE the special step in and step out icons are used to indicate that the task has entered or left a service routine procedure.
Cell context menu	Contains options that change the attribute value representation and update commands. The first options are available only for integer attributes (not a String attribute) and they affect the attribute representation globally. This means that the change is saved into the model and all views will show that attribute consistently. A subgroup (Show label, Show value, Show label and value) of these options treat only enum attributes and handle label-value show. The other subgroup controls the base number representation (Binary, Octal, Decimal, Hex). The options from update group are: Toggle update which bypass the renew policies set to attribute column or row and Update which refreshes unconditionally the attribute's value direct from the target and not from model's.

Trace

This tab is designated to show a simple trace view of the traceable attributes. You can set the sample time and the trace buffer length. Because of a **System Browser** constraint this view illustrated trace results only when the core is stopped; for example. after a step.

Figure 4.22 System Browser View — Trace



OSEK
Tasks
Implementation
Trace
type filter text
OS
os1
TASK
TASKCNT
TASKRCV1
TASKRCV2
TASKSND1
TASKSND2
TASKSTOP
ALARM
STOPALARM
TIMLIMITALARM
MESSAGECONTAINER
MsgAReceived
MsgBReceived
RESOURCE
MSGAACCESS
MSGBACCESS
STACK
ISR_STACK
MAIN_STACK
TASKRCV1_STACK
vs_COUNTER
Binary
Octal
● Decimal
Hex
Toggle update
Refresh
Show label
Show value
● Show label and value

Application:HCS08, sample_osek_qe128.abs, Implementation:ORTI_OSEKturbo_OS08_2_2_1_86 (KOIL ver. 2.1, Semantics "ORTI" ver. 2.1)

Update attributes
☑ PRIORITY ☑ STATE ☑ vs_EVENTS ☑ vs_WAITEVENTS ☑ STACK

	Name	PRIORITY	STATE	vs_EVE...	vs_WAITEVENTS	vs_MASKS	STACK	vs_PROPERTY
☑	TASKCNT	2 '1'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , NONPREEMPT
☑	TASKRCV1	6 '5'	0 'RUNNING'	1	0	MSGAEVENT = 0001	??=341	EXTENDED, FULLPREE
☑	TASKRCV2	5 '4'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSND1	4 '3'	2 'READY'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSND2	3 '2'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , FULLPREEMPT
☑	TASKSTOP	1 '0'	3 'SUSPENDED'	0	0		⚠ 0 'NO_STACK'	BASIC , NONPREEMPT

```
TASKSTOP :: STATE object attribute
Static: No
Formula: ( OsRunning != 6 ) * ( (                 & 0x04 ) != 0 ) * 2 + ( OsTaskStatus[6] == 0 ) * 3
Runtime value: 3 'SUSPENDED'

Attribute STATE description:
Task State
Type: Enum
Traced:No
C-Type: unsigned char
Enum mapping:
  RUNNING = 0 (0)
```

Exceptions

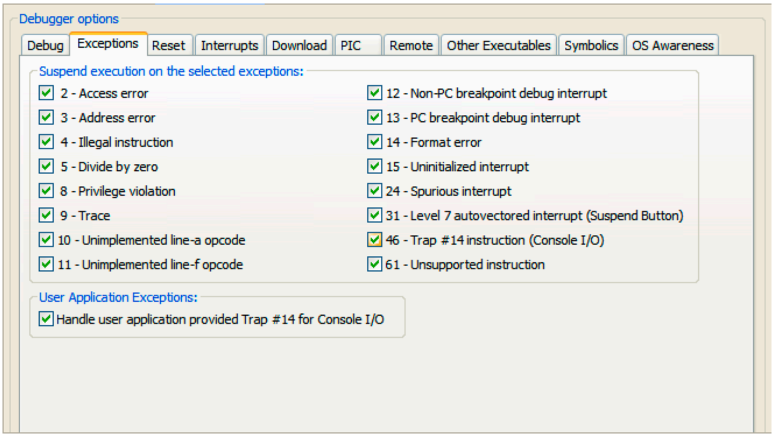
The **Exceptions** tab (Figure 4.23) is available with P&E Microcomputer Systems, simulator, and Freescale USB and Ethernet TAP remote connections. Use this tab to specify hardware exceptions that you want the debugger to catch. Before you load and run the program, the debugger inserts its own exception vector for each exception you check in tab. To use your own exception vectors instead, clear the corresponding checkboxes.

If you check any options, the debugger reads the Vector_Based_Register (VBR), finds the corresponding existing exception vector and then writes a new vector at that register location. The address of this new vector is offset 0x408 from the VBR address. For example, if the VBR address is 0x0000 0000, the new vector at address 0x0000 0408 catches and handles the checked exceptions.

The debugger writes a Halt instruction and a Return from Exception instruction at this same location.

NOTE If your exceptions are in Flash or ROM, do not check any boxes in the **CF Exceptions** panel.

Figure 4.23 Exceptions Tab — For ColdFire



Suspend execution on the selected exceptions:

- ☑ 2 - Access error
- ☑ 3 - Address error
- ☑ 4 - Illegal instruction
- ☑ 5 - Divide by zero
- ☑ 8 - Privilege violation
- ☑ 9 - Trace
- ☑ 10 - Unimplemented line-a opcode
- ☑ 11 - Unimplemented line-f opcode
- ☑ 12 - Non-PC breakpoint debug interrupt
- ☑ 13 - PC breakpoint debug interrupt
- ☑ 14 - Format error
- ☑ 15 - Uninitialized interrupt
- ☑ 24 - Spurious interrupt
- ☑ 31 - Level 7 autovectored interrupt (Suspend Button)
- ☑ 46 - Trap #14 instruction (Console I/O)
- ☑ 61 - Unsupported instruction

User Application Exceptions:

- ☑ Handle user application provided Trap #14 for Console I/O

Table 4.20 describes the exceptions settings.

Table 4.20 Exceptions Settings — For ColdFire

Option	Description
2 - Access Error	Determines whether the debugger handles the access error exception. Check this option to catch and display access errors. Clear this option to ignore access errors.
3 - Address error	Determines whether the debugger handles the address error exception. Check this option to catch and display address errors. Clear this option to ignore address errors.
4 - Illegal instruction	Determines whether the debugger handles an invalid instruction exception. Check this option to catch and display invalid instructions. Clear this option to ignore invalid instructions.

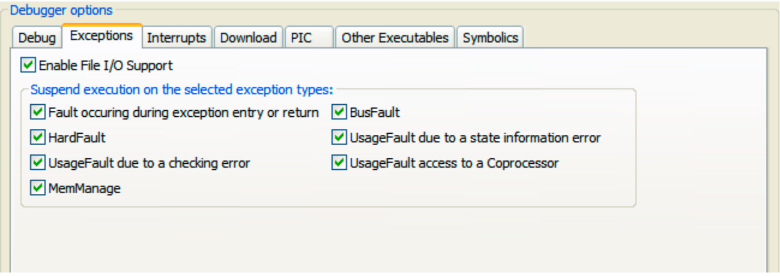
Table 4.20 Exceptions Settings — For ColdFire (*continued*)

Option	Description
5 - Divide by zero	Determines whether the debugger handles a divide by zero exception. Check this option to catch and display any attempt to divide by zero. Clear this option to ignore divide by zero attempts.
8 - Privilege Violation	Determines whether the debugger handles a privilege violation exception. Check this option to catch and display privilege violations. Clear this option to ignore privilege violations.
9 - Trace	Determines whether the debugger handles a Trace exception. Check this option to catch and display trace exceptions. Clear this option to ignore trace exceptions.
10 - Unimplemented line-a opcode	Determines whether the debugger handles a unimplemented line-A opcode exception. Check this option to catch and display unimplemented line-A opcodes Clear this option to ignore unimplemented line-A opcodes.
11- Unimplemented line-f opcode	Determines whether the debugger handles a unimplemented line-F opcode exception. Check this option to catch and display unimplemented line-F opcodes. Clear this option to ignore unimplemented line-F opcodes.
12 - Non-PC breakpoint debug interrupt	Determines whether the debugger handles non-PC breakpoint debug interrupts. Check this option to catch and display non-PC breakpoint debug interrupts. Clear this option to ignore non-PC breakpoint debug interrupts.
13 - PC breakpoint debug interrupt	Determines whether the debugger handles PC breakpoint debug interrupts. Check this option to catch and display PC breakpoint debug interrupts. Clear this option to ignore PC breakpoint debug interrupts.

Table 4.20 Exceptions Settings — For ColdFire (*continued*)

Option	Description
14 - Format error	Determines whether the debugger handles format error exceptions. Check this option to catch and display format errors. Clear this option to ignore format errors.
15 - Uninitialized interrupt	Determines whether the debugger handles uninitialized interrupts. Check this option to catch and display uninitialized interrupts. Clear this option to ignore uninitialized interrupts.
24 - Spurious interrupt	Determines whether the debugger handles spurious interrupts. Check this option to catch and display spurious interrupts. Clear this option to ignore spurious interrupts.
31 - Level 7 autovectored interrupt (Suspend Button)	Determines whether the debugger handles level 7 suspend button exceptions. Check this option to catch and display the use of the level 7 interrupts. Clear this option to ignore level 7 interrupts.
46 - Trap #14 instruction (Console I/O)	Determines whether the debugger handles trap # 14 instructions, which implement console I/O. Clear this option to ignore trap 14 instructions. Check this option to catche and display uses of trap 14 instructions.
61 - Unsupported instruction	Determines whether the debugger handles the unsupported instruction exception. Check this options to catch and display unsupported instructions. Clear this option to ignore unsupported instructions.
Handle user application provided Trap #14 for console I/O	Determines whether the debugger handles trap # 14 exceptions when they occur in an application. Clear this option to ignore trap 14 instructions. Check this option to catches and display uses of trap 14 instructions.

Figure 4.24 Exceptions Tab — For Kinetis



☑ Enable File I/O Support

Suspend execution on the selected exception types:

☑ Fault occuring during exception entry or return
☑ HardFault
☑ UsageFault due to a checking error
☑ MemManage
☑ BusFault
☑ UsageFault due to a state information error
☑ UsageFault access to a Coprocessor

Table 4.21 describes the exceptions settings.

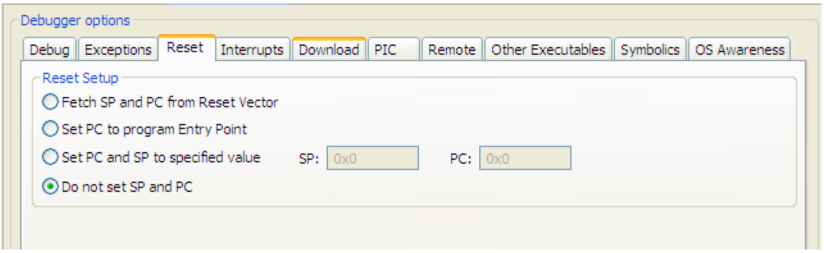
Table 4.21 Exceptions Settings — For Kinetis

Option	Description
Enable File I/O Support	
Fault occuring during exception entry or return	
Hard Fault	
UsageFault due to checking error	
MemManage	
BusFault	
UsageFault due to state information error	
UsageFault access to a Coprocessor	

Reset

The **Reset** tab (Figure 4.25) specifies the setup actions that the microcontroller takes when it comes out of a reset.

Figure 4.25 Reset Tab



○ Fetch SP and PC from Reset Vector
○ Set PC to program Entry Point
○ Set PC and SP to specified value SP: 0x0 PC: 0x0
● Do not set SP and PC

Table 4.22 Reset Tab

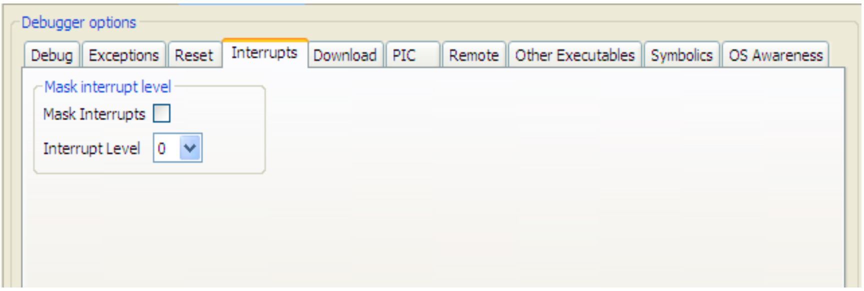
Option	Description
Fetch SP and PC from Reset Vector	When selected, the debugger fetches the base of the stack and the start address from the reset vector, and loads them into the Microcontroller's SP and PC, respectively. Used for ROM build targets.
Set PC to program Entry Point	When selected, the debugger sets the Microcontroller's PC to the program's start address. Used to emulate reset for RAM build targets.
Set PC and SP to specified value	When selected, the debugger takes the user-supplied values for SP and PC and loads them the corresponding Microcontroller registers. Used to reference the entry point of a boot loader. When selected, the `SP:` and `PC:` text entry boxes are active. Enter the hexadecimal addresses for SP and PC in these boxes.
Do not set SP or PC	When selected, the debugger takes no action and the Microcontroller uses the default addresses in the PC and SP.

Interrupts

Debugging an application involves single-stepping through code. However, if you do not modify the behavior of interrupts that are part of normal code execution, an interrupt may occur and the debugger jumps to the interrupt handler code, rather than single-stepping to the next instruction. Therefore, you must mask, or inhibit, certain interrupt levels to prevent the interrupts from happening. The interrupt levels that you inhibit varies, depending upon the microcontroller.

Use this tab to inhibit or allow interrupts. When inhibiting interrupts, you can mask interrupts below a level that you specify. Figure 4.26 shows the Interrupts tab.

Figure 4.26 Interrupts Tab



Mask Interrupts ☐

Interrupt Level 0

Table 4.23 explains each option.

Table 4.23 Interrupts Tab

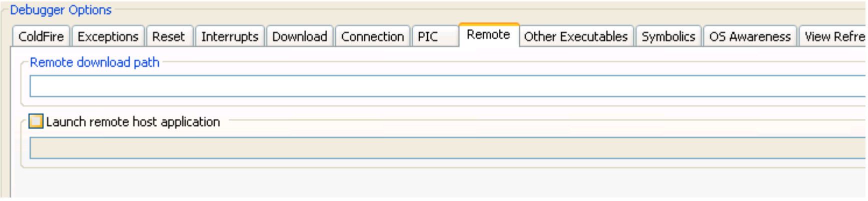
Option	Description
Mask Interrupts	Determines whether the debugger inhibits or allows interrupts. Check this option to inhibit interrupts, using the level specified in the `Interrupt Level` option. Uncheck this option to permit all interrupts.
Interrupt Level	Use this option to specify the interrupt level that the debugger inhibits. Level 0 corresponds to the lowest priority interrupt, while level 7 is the highest. If you specify a level of 4, then the debugger inhibits interrupts of level 0 through 4, while interrupts at levels 5 through 7 execute.

NOTE The exact definitions of interrupt levels are different for each target microcontroller, and masking all interrupts can cause erratic behavior. This means that finding the best interrupt level to mask can involve trial and error. Be alert for any code statements that change the interrupt mask; stepping over such a statement can modify the settings in the tab.

Remote

When debugging a Linux application, use this tab to specify where the debugger downloads the program for debug on the Linux host system, and whether to launch any optional applications while debugging. Figure 4.27 shows the **Remote** tab view.

Figure 4.27 Remote Tab



Remote download path

☐ Launch remote host application

Table 4.24 describes the **Remote** tab settings.

Table 4.24 Remote Tab Settings

Option	Description
Remote download path	Specifies the directory path that the debugger downloads the test program into.
Launch remote host application	Specifies the directory path of a Linux program that is to be launched along with the test program.

EPPC Exceptions

Use this page to specify which processor exceptions you want the debugger to catch.

Table 4.25 describes the **EPPC Exceptions** tab settings.

Table 4.25 EPPC Exceptions Tab Settings

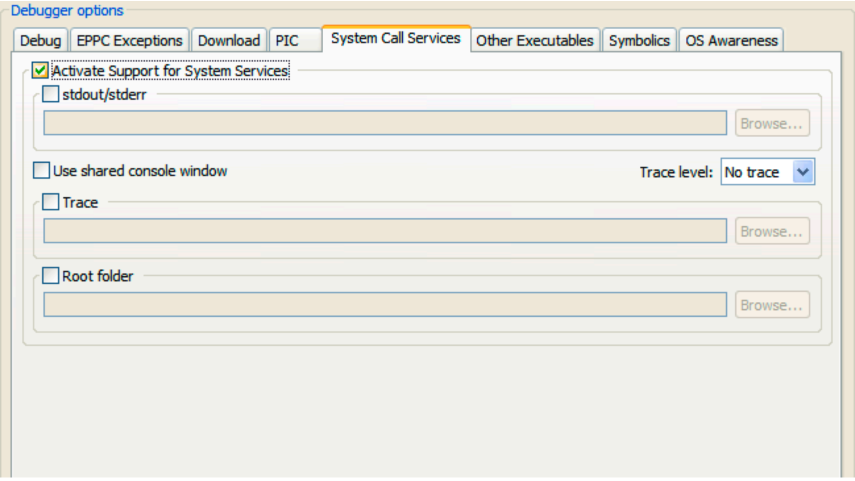
Option	Description
The features of this page view are currently not supported by this implementation.	

System Call Services

Use this page to activate the debugger's support for system calls and to select options that define how the debugger handles system calls. The CodeWarrior debugger provides system call support over JTAG. System call support lets bare board applications use the functions of host OS service routines. This feature is useful if you do not have a board support package (BSP) for your target board. The host debugger implements these services. Therefore, the host OS service routines are available only when you are debugging a program on a target board or simulator.

NOTE The OS service routines provided must comply with an industry-accepted standard. The definitions of the system service functions provided are a subset of Single UNIX Specification (SUS).

Figure 4.28 Debug Configurations — System Call ServicesTab



☑ Activate Support for System Services
☐ stdout/stderr
☐ Use shared console window
Trace level: No trace
☐ Trace
☐ Root folder

Table 4.26 describes the settings on the **System Call Services** panel.

Table 4.26 System Call Services Settings

Option	Description
Activate Support for System Services	Check this option to enable support for system services. All the other options on the **System Call Services** panel are enabled only if you check this checkbox.
stdout/stderr	By default, the output written to stdout and stderr appears in a CodeWarrior IDE “console” window. To redirect console output to a file, check the stdout/stderr checkbox. Click **Browse** to display a dialog box and specify the path and name of this file.

Table 4.26 System Call Services Settings

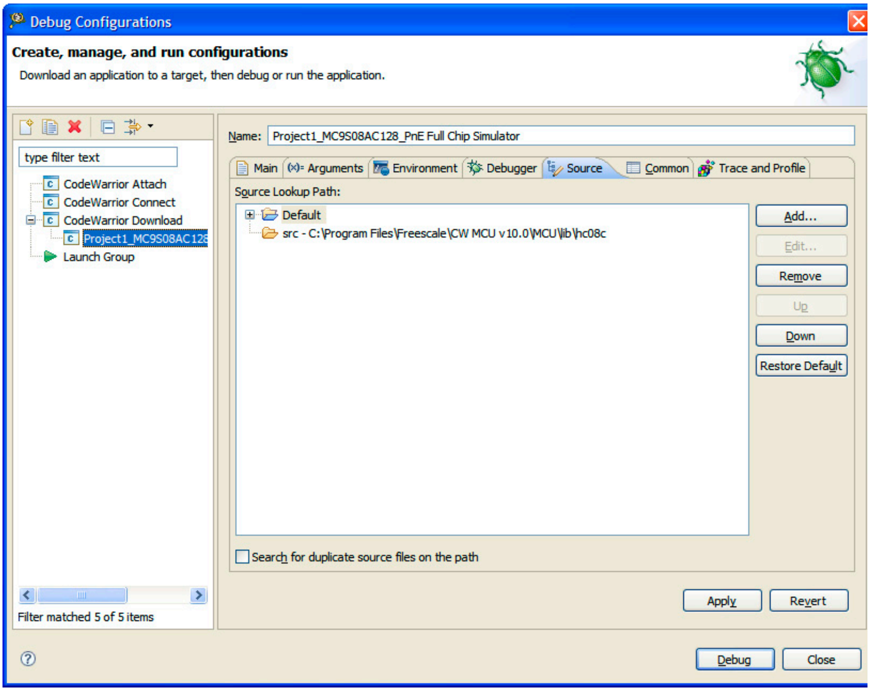
Option	Description
Use shared console window	Check this option if you wish to share the same console window between different debug targets. This setting is useful in multi-core or multi-target debugging.
Trace level	Use this dropdown list to specify the system call trace level. The place where the debugger displays the traced system service requests is determined by the Trace checkbox. The system call trace level options available are: • **No Trace** — system calls are not traced • **Summary** — the requests for system services are displayed • **Detailed** — the requests for system services are displayed along with the arguments/parameters of the request
Trace	By default, traced system service requests appear in a CodeWarrior IDE “console” window. To log traced system service requests to a file, check the Trace checkbox. Click **Browse** to display a dialog box and define the path and name of this file. In a project created by the New Project wizard, use the library syscall.a rather than a UART library for handling the output.
Root folder	The directory on the host system which contains the OS routines that the bare board program uses for system calls.

Source

Use this page to specify the location of source files used when debugging a C or C++ application. By default, this information is taken from the build path of your project.

The **Source** tab options are explained in Table 4.27.

Figure 4.29 Debug Configurations — Source Tab



Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Project1_MC9S08AC128_PnE Full Chip Simulator

Source Lookup Path:

- Default
- src - C:\Program Files\Freescale\CW MCU v10.0\MCU\lib\hc08c

☐ Search for duplicate source files on the path

Filter matched 5 of 5 items

Debug Configurations
type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Project1_MC9S08AC128
Launch Group
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Add...
Edit...
Remove
Up
Down
Restore Default
Apply
Revert
Debug
Close

Table 4.27 Source Tab Options

Option	Description
Source Lookup Path	Lists the source paths used to load an image after connecting the debugger to the target.
Add	Click to add new source containers to the Source Lookup Path search list.
Edit	Click to modify the content of the selected source container.
Remove	Click to remove selected items from the **Source Lookup Path** list.
Up	Click to move selected items up the **Source Lookup Path** list.
Down	Click to move selected items down the **Source Lookup Path** list.
Restore Default	Click to restore the default source search list.

Table 4.27 Source Tab Options (*continued*)

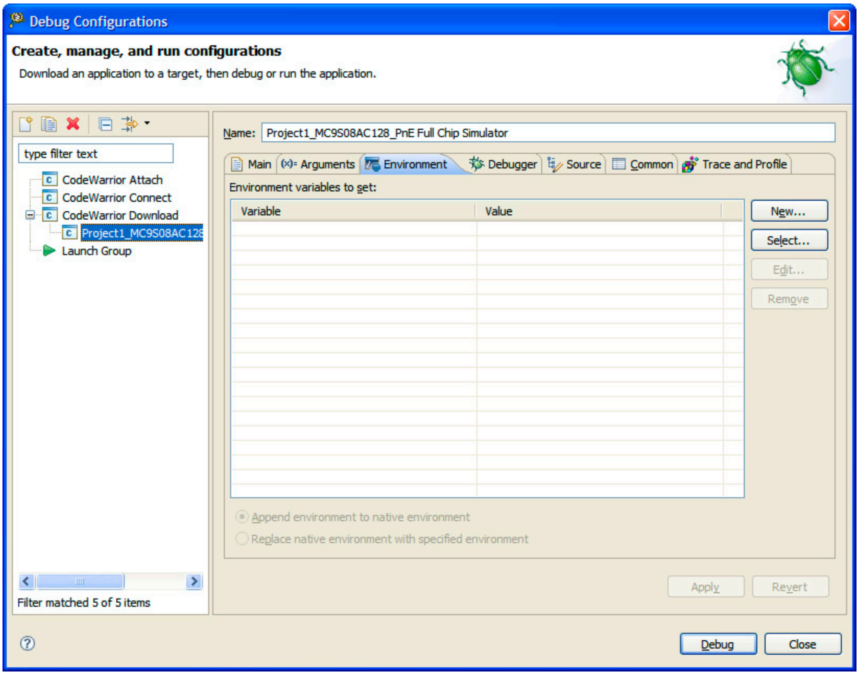
Option	Description
Search for duplicate source files on the path	Select to search for files with the same name on a selected path.

Environment

Use this page to specify the environment variables and values to use when an application runs.

The **Environment** tab options are explained in Table 4.28.

Figure 4.30 Debug Configurations — Environment Tab



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Project1_MC9S08AC128_PnE Full Chip Simulator

Environment variables to set:

Variable	Value

● Append environment to native environment

○ Replace native environment with specified environment

Filter matched 5 of 5 items

type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Project1_MC9S08AC128
Launch Group
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
New...
Select...
Edit...
Remove
Apply
Revert
Debug
Close

Table 4.28 Environment Tab Options

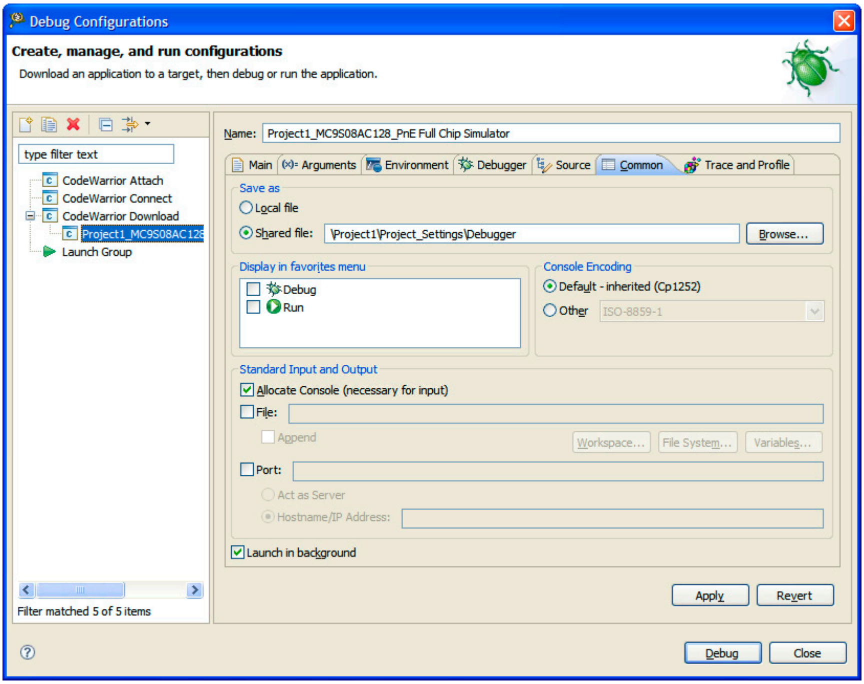
Option	Description
Environment Variables to set	Lists the environment variable name and its value.
New	Click to create a new environment variable.
Select	Click to select an existing environment variable.
Edit	Click to modify the name and value of a selected environment variable.
Remove	Click to remove selected environment variables from the list.
Append environment to native environment	Select to append the listed environment variables to the current native environment.
Replace native environment with specified environment	Select to replace the current native environment with the specified environment set.

Common

Use this page to specify the location to store your run configuration, standard input and output, and background launch options.

The **Common** tab options are explained in Table 4.29.

Figure 4.31 Debug Configurations — Common Tab



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Project1_MC9S08AC128_PnE Full Chip Simulator

Save as

○ Local file

● Shared file: \Project1\Project_Settings\Debugger

Display in favorites menu

☐ Debug

☐ Run

Console Encoding

● Default - inherited (Cp1252)

○ Other ISO-8859-1

Standard Input and Output

☑ Allocate Console (necessary for input)

☐ File:

☐ Append

☐ Port:

○ Act as Server

● Hostname/IP Address:

☑ Launch in background

Filter matched 5 of 5 items

type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Project1_MC9S08AC128
Launch Group
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Browse...
Workspace...
File System...
Variables...
Apply
Revert
Debug
Close

Table 4.29 Common Tab Options

Option	Description
Local file	Select to save the launch configuration locally.
Shared file	Select to specifies the path of, or browse to, a workspace to store the launch configuration file, and be able to commit it to a repository.
Display in favourites menu	Check to add the configuration name to Run or Debug menus for easy selection.
Console Encoding	Select an encoding scheme to use for console output.
Allocate Console (necessary for input)	Select to assign a console view to receive the output.
File	Specify the file name to save output.

Table 4.29 Common Tab Options (*continued*)

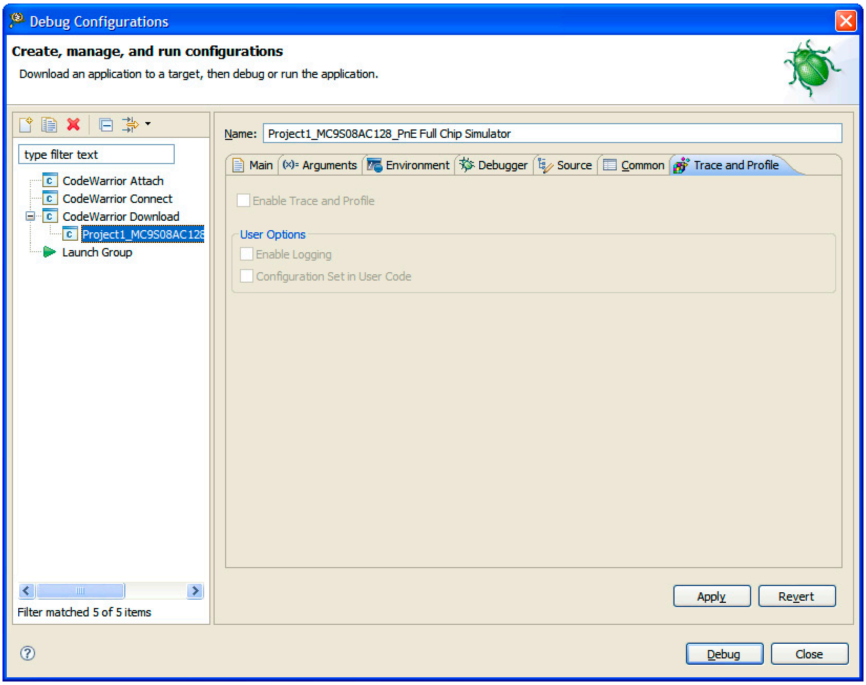
Option	Description
Browse Workspace	Specifies the path of, or browse to, a workspace to store the output file.
Browse File System	Specifies the path of, or browse to, a file system directory to store the output file.
Variables	Select variables by name to include in the output file.
Append	Check to append output. Uncheck to recreate file each time.
Port	Check to redirect standard output (`stdout`, `stderr`) of a process being debugged to a user specified socket. **Note:** You can also use the `redirect` command in debugger shell to redirect standard output streams to a socket.
Act as Server	Select to redirect the output from the current process to a local server socket bound to the specified port.
Hostname/ IP Address	Select to redirect the output from the current process to a server socket located on the specified host and bound to the specified port. The debugger will connect and write to this server socket via a client socket created on an ephemeral port
Launch in background	Check to launch configuration in background mode.

Trace and Profile

Use this page to configure the selected launch configuration for simulator and hardware profiling.

The **Trace and Profile** tab options are explained in Table 4.30.

Figure 4.32 Debug Configurations — Trace and Profile Tab



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Project1_MC9S08AC128_PnE Full Chip Simulator

☐ Enable Trace and Profile

User Options

☐ Enable Logging

☐ Configuration Set in User Code

Filter matched 5 of 5 items

type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Project1_MC9S08AC128
Launch Group
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Apply
Revert
Debug
Close

Table 4.30 Trace and Profile Tab Options

Option	Description
Enable Trace and Profile	Check to collect trace and profiling data. This option is unchecked by default.
Select the analysis type	Specify the type of analysis. **Simulator based analysis** — Select to perform an analysis based on simulator. **Note:** This option is available only when you are working on a Simulator project. **Hardware based analysis (VTB)** — Select to perform an analysis based on hardware **Instrumented code analysis (in target profiler)** — Select to perform an analysis based on instrumented code. **Note:** Only code coverage information is reported

Table 4.30 Trace and Profile Tab Options (*continued*)

Option	Description
Exclude internal libraries from performance statistics	Check to exclude the internal (runtime) library functions from the performance statistics. Else, the performance statistics includes the internal (runtime) library functions.
Separate interrupts from functions	Check to profile the interrupts separately from functions. For more details, see *Performance Data*.
Backup ELD after measurement	Check to backup the target image files.
Auto Save Results	Check to save the previously captured result, automatically at the start of each run.
Trace Offload Method	
JTAG	Select to collect data from the board memory and send it to the computer. Ethernet and USB JTAG probe are supported. Ethernet TAP probe has a 16-pin connector, which automatically supports target system signal levels from 1.8V to 3.3V.
HSST (High Speed Simultaneous Transfer)	Select to collect data from the board memory and send it to the computer. HSST is faster than JTAG. If you are using MSC8144 target and VTB small buffer, it is 12.82 times faster than JTAG. **Note**: Select **Write Mode as Trace Event Request** and also set the address of the **Trace Event Request Address** before selecting the HSST option.
SmartDSP HEAT (Host Exchange over Asynchronous Transfer)	Select to collect data from board memory and send it to the computer using a GigaEthernet connection. If you are using MSC8144 target and VTB small buffer, SmartDSP HEAT is 598.7 times faster than JTAG and 46.7 times faster than HSST.
Custom	Select to customize the trace output. In the current release, the functionality of this option is similar to JTAG.
Trace Configuration Method	
Auto - settings done by CW	Select this option if the settings specified on Trace and Profile page is used to collect trace and profiling data.

Table 4.30 Trace and Profile Tab Options (*continued*)

Option	Description
Manual - settings done by user code	Select to configure trace in the code without using the Trace and Profile page of Debug launch. Ensure that all DPU and OCE registers are configured correctly
Counted Events	
Count cycles using OCE	In this mode, only the OCE cycle counter is used and the trace will contain source, destination address, and value of the cycle counter. If this radio button is selected, the Trace Mode of the DPU settings is automatically set to OCE Trace - no compression in the Advanced Settings panel
Use DPU Counters	In this mode, 6 DPU counters are used which count the events specified by one of the predefined configurations. If this radio button is selected, the Trace Mode of the DPU settings is automatically set to DPU Counters in the Advanced Settings panel
DPU Configuration Settings	Select to use a predefined counters configuration. If you want to customize the setup, choose the custom option and configure the counters from the Advanced Settings panel
Advanced Settings	Click to open the **Preferences** dialog box and configure the DPU, OCE, VTB, HSST, and SmartDSP HEAT settings.

Debugging Bare Board Software

This topic applies to debugging software on bare board systems, that is, for hardware that is not running an operating system.

The topics are:

- Displaying Register Contents
- Using Register Details Window
- Setting Watchpoints
- Removing Watchpoints
- Setting Breakpoints
- Removing Breakpoints
- Setting Stack Crawl Depth
- Hard Resetting

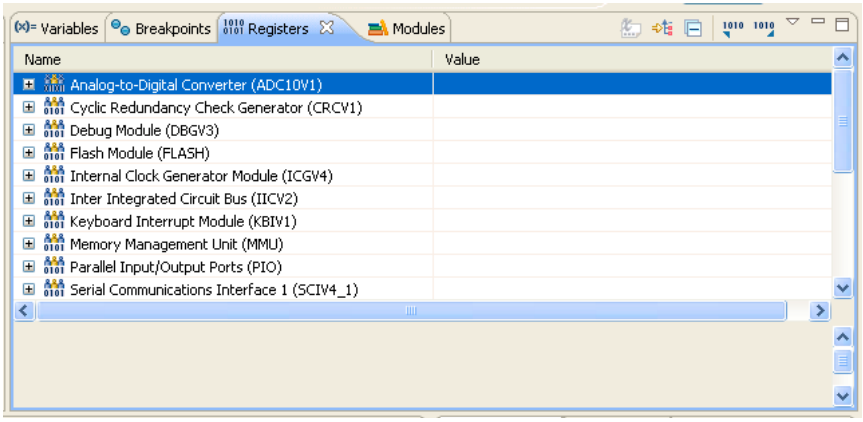
Displaying Register Contents

Use the **Registers** view to display and modify the contents of the registers of the processor on your target board. To display this view from the **Debug** perspective, Select **Window > Show View > Registers**, and the **Registers** view appears.

The **Registers** view displays categories of registers in a tree format. To display the contents of a particular category of registers, expand the tree element of the register category of interest. Figure 4.33 shows the **Registers** view with the General Purpose Registers tree element expanded.

TIP You can also view and update registers by issuing the `reg`, `change`, and `display` commands in the CodeWarrior **Debugger Shell** view.

Figure 4.33 Registers View



Name	Value
Analog-to-Digital Converter (ADC10V1)	
Cyclic Redundancy Check Generator (CRCV1)	
Debug Module (DBGV3)	
Flash Module (FLASH)	
Internal Clock Generator Module (ICGV4)	
Inter Integrated Circuit Bus (IICV2)	
Keyboard Interrupt Module (KBIV1)	
Memory Management Unit (MMU)	
Parallel Input/Output Ports (PIO)	
Serial Communications Interface 1 (SCIV4_1)	

Adding Register Group

By default, the **Registers** view lists the related register groups in a tree structure. You can add a custom group of registers to the default tree structure. To add a new register group:

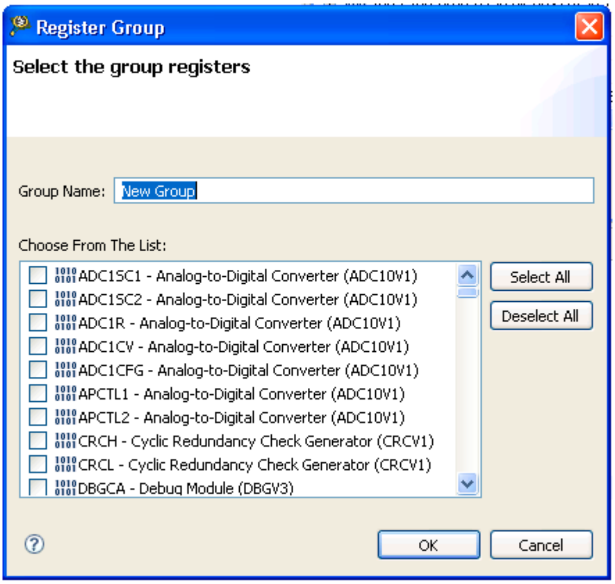
1. Right-click in the **Registers** view.

   A context menu appears.

2. Select **Add Register Group**.

   The **Register Group** dialog box appears (Figure 4.34).

Figure 4.34 Register Group Dialog Box



Register Group

Select the group registers

Group Name: New Group

Choose From The List:

☐ ADC1SC1 - Analog-to-Digital Converter (ADC10V1)
☐ ADC1SC2 - Analog-to-Digital Converter (ADC10V1)
☐ ADC1R - Analog-to-Digital Converter (ADC10V1)
☐ ADC1CV - Analog-to-Digital Converter (ADC10V1)
☐ ADC1CFG - Analog-to-Digital Converter (ADC10V1)
☐ APCTL1 - Analog-to-Digital Converter (ADC10V1)
☐ APCTL2 - Analog-to-Digital Converter (ADC10V1)
☐ CRCH - Cyclic Redundancy Check Generator (CRCV1)
☐ CRCL - Cyclic Redundancy Check Generator (CRCV1)
☐ DBGCA - Debug Module (DBGV3)

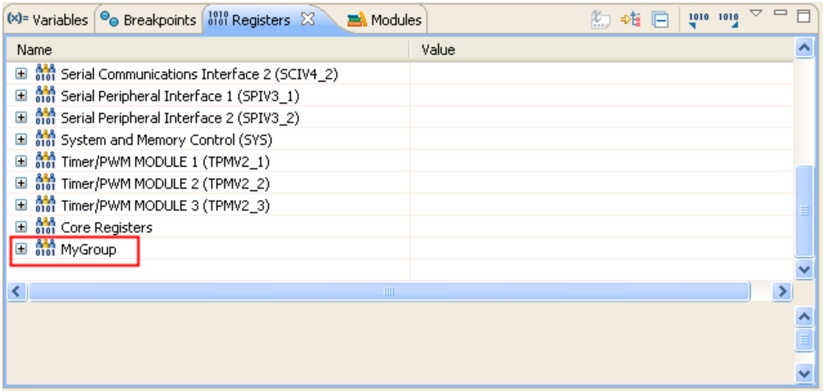
3. Enter in the **Group Name** text box a descriptive name for the new group. For example, *MyGroup*.
4. Check the checkbox adjacent to each register you want to add in the new group.

TIP Click **Select All** to check all of the checkboxes. Click **Deselect All** to clear all the checkboxes.

5. Click **OK**.

   The **Register Group** dialog box closes. The new group name appears in the **Registers** view.

Figure 4.35 New Register Group



Name	Value
Serial Communications Interface 2 (SCIV4_2)	
Serial Peripheral Interface 1 (SPIV3_1)	
Serial Peripheral Interface 2 (SPIV3_2)	
System and Memory Control (SYS)	
Timer/PWM MODULE 1 (TPMV2_1)	
Timer/PWM MODULE 2 (TPMV2_2)	
Timer/PWM MODULE 3 (TPMV2_3)	
Core Registers	
MyGroup	

Editing Register Group

In the **Registers** view, you can edit both the default register groups and the groups that you add. To do so:

1. In the **Register** view, right-click on the name of the register group you want to edit. For example, right-click on *MyGroup*.

   A context menu appears.

2. Select **Edit Register Group**.

   The **Register Group** dialog box appears (Figure 4.34).

3. If required, enter a new name for the group in the **Group Name** text box.
4. Check the checkbox adjacent to each register you want to add in the group.

TIP Click **Select All** to check all of the checkboxes. Click **Deselect All** to clear all the checkboxes.

5. Click **OK**.

   The **Register Group** dialog box closes. The new group name appears in the **Registers** view.

Removing Register Group

In the **Registers** view, you can remove register groups. To remove a register group:

1. In the **Registers** view, right-click on the register group you want to remove.

   A context menu appears.

2. Select **Remove Register Group**.

   The selected register group is removed from the **Registers** view.

Changing Register's Bit Value

To change a bit value in a register, first switch the IDE to the **Debug** perspective, start a debugging session and perform these steps.

1. Open the **Registers** view by selecting **Window > Show View > Registers**.
2. Expand the register group that contains the register with the bit value that you want to change.
3. Click on the register's current bit value in the view's **Value** column.

   The value appears editable.
4. Type in the new value.
5. Press the **Enter** key.

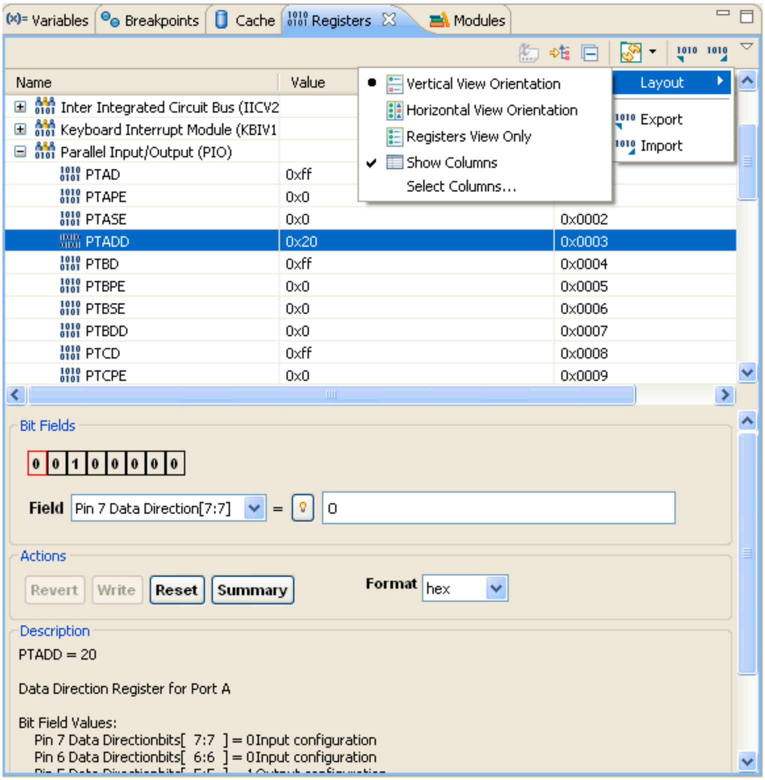
   The debugger updates the bit value. The bit value in the **Value** column changes to reflect your modification.

NOTE Modified values are highlighted in yellow.

Using Register Details Window

The default state of the **Registers** view is to provide details on the processor's registers (Figure 4.36).

Figure 4.36 Register View — Detailed Information


(x)= Variables
Breakpoints
Cache
Registers
Modules
● Vertical View Orientation
Horizontal View Orientation
Registers View Only
✓ Show Columns
Select Columns…
Layout
Export
Import
Bit Fields
Actions
Revert
Write
Reset
Summary
Description

Name	Value	
⊞ Inter Integrated Circuit Bus (IICV2		
⊞ Keyboard Interrupt Module (KBIV1		
⊟ Parallel Input/Output (PIO)		
PTAD	0xff	
PTAPE	0x0	
PTASE	0x0	0x0002
PTADD	0x20	0x0003
PTBD	0xff	0x0004
PTBPE	0x0	0x0005
PTBSE	0x0	0x0006
PTBDD	0x0	0x0007
PTCD	0xff	0x0008
PTCPE	0x0	0x0009

0	0	1	0	0	0	0	0

Field Pin 7 Data Direction[7:7] = 0

Format hex

PTADD = 20

Data Direction Register for Port A

Bit Field Values:
Pin 7 Data Directionbits[7:7] = 0Input configuration
Pin 6 Data Directionbits[6:6] = 0Input configuration

The **Registers** view displays several types of register details:

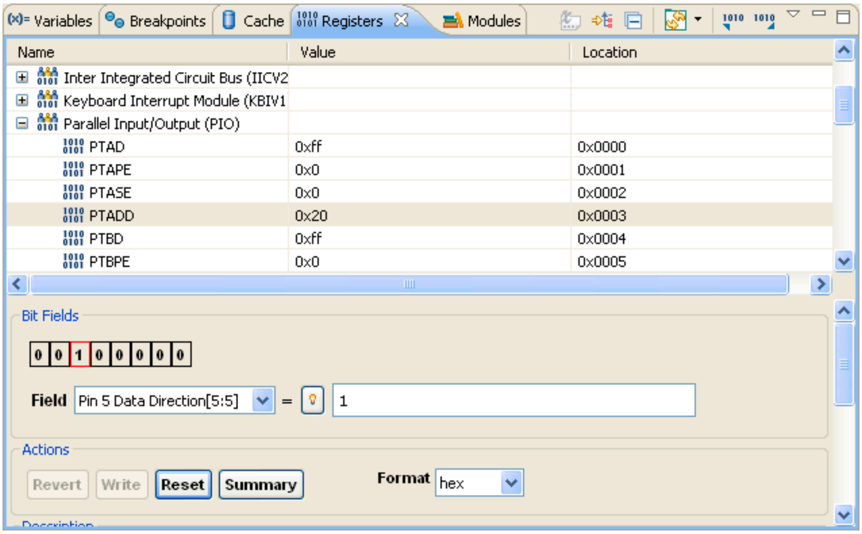
- Bit Fields
- Description
- Actions

NOTE To display the register details, first you have to select a register, then expand the view by clicking and dragging the areas at the bottom of the **Registers** view to reveal the Bit Field, Description, and Actions portions of the view.

Bit Fields

The Bit Fields group of the **Registers** view (Figure 4.37) shows a graphical representation of the selected register's bit values. This graphical representation shows how the register organizes bits. You can use this representation to select and change the register's bit values. Hover the cursor over each part of the graphical representation in order to see additional information.

Figure 4.37 Register Details — Bit Fields Group



Name	Value	Location
Inter Integrated Circuit Bus (IICV2		
Keyboard Interrupt Module (KBIV1		
Parallel Input/Output (PIO)		
PTAD	0xff	0x0000
PTAPE	0x0	0x0001
PTASE	0x0	0x0002
PTADD	0x20	0x0003
PTBD	0xff	0x0004
PTBPE	0x0	0x0005

Bit Fields

0	0	1	0	0	0	0	0

Field Pin 5 Data Direction[5:5] = 1

Actions

Format hex

TIP You can also view register details by issuing the `reg` command in the **Debugger Shell** view.

A *bit field* is either a single bit or a collection of bits within a register. Each bit field has a mnemonic name that identifies it. You can use the **Field** list box to view and select a particular bit field of the selected register. The list box shows the mnemonic name and bit-value range of each bit field. In the Bit Fields graphical representation, a box surrounds each bit field. A red box surrounds the bit field shown in the **Field** list box.

After you use the **Field** list box to select a particular bit field, you see its current value in the **=** text box. If you change the value shown in the text box, the **Registers** view shows the new bit-field value.

The minimum resolution of bit-field descriptions is 2 bits. Consequently, register details are not available for single-bit overflow registers.

The maximum resolution of bit-field descriptions is 32 bits.

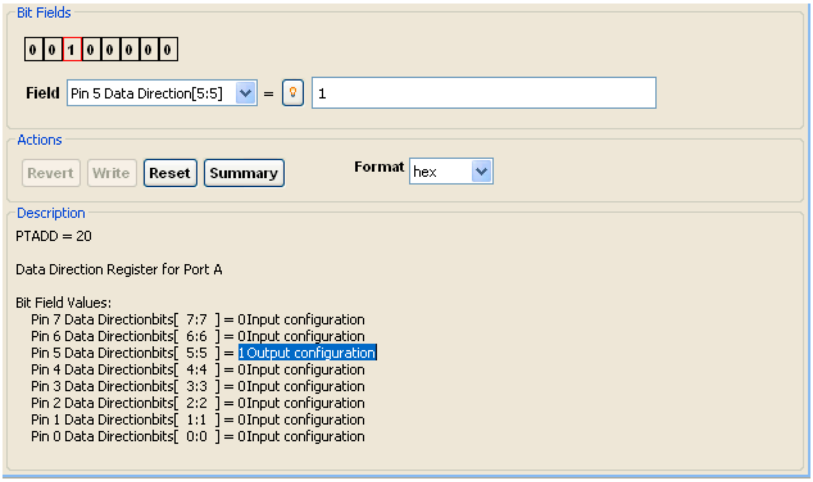
Changing Bit Field

To change a bit field in a register, you must first start a debugging session and then open the **Registers** view.

To change a bit field, perform these steps.

1. In the **Registers** view, view register details.
2. Expand the register group that contains the bit field you want to change.
3. Register details appear (Figure 4.38) in the **Registers** view.

Figure 4.38 Registers View — Register Details



0 0 1 0 0 0 0 0

Field Pin 5 Data Direction[5:5] = 1

Format hex

PTADD = 20

Data Direction Register for Port A

Bit Field Values:

```
   Pin 7 Data Directionbits[  7:7  ] = 0Input configuration
   Pin 6 Data Directionbits[  6:6  ] = 0Input configuration
   Pin 5 Data Directionbits[  5:5  ] = 1Output configuration
   Pin 4 Data Directionbits[  4:4  ] = 0Input configuration
   Pin 3 Data Directionbits[  3:3  ] = 0Input configuration
   Pin 2 Data Directionbits[  2:2  ] = 0Input configuration
   Pin 1 Data Directionbits[  1:1  ] = 0Input configuration
   Pin 0 Data Directionbits[  0:0  ] = 0Input configuration
```

4. From the expanded register group above the register details, select the name of the register that contains the bit field that you want to change.

   The **Bit Fields** group displays a graphical representation of the selected bit field. The **Description** group displays explanatory information about the selected bit field and parent register.

5. In the **Bit Fields** group, click the bit field that you want to change. Alternatively, use the **Field** list box to specify the bit field that you want to change.
6. In the **=** text box, type the new value that you want to assign to the bit field.
7. In the **Action** group, click the **Write** button.

NOTE The **Revert** and **Write** buttons appear enabled if the value in the **=** field is changed or you reset the values.

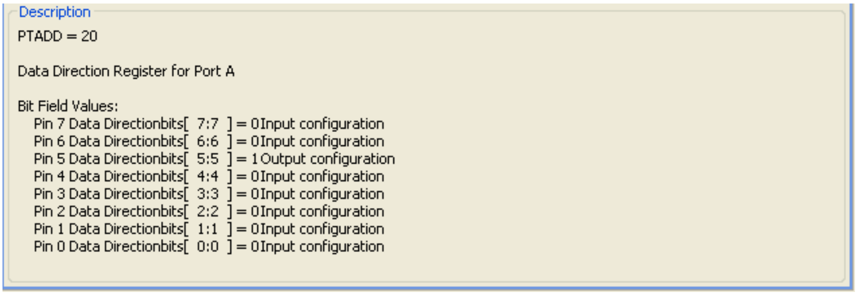
The debugger updates the bit-field value. The bit values in the **Value** column and the **Bit Fields** group change to reflect your modification.

NOTE Click the **Reset** button to discard your changes and restore the original bit-field value. Click the **Revert** button to revert to the last changes made.

Description

The Description group of the **Registers** view (Figure 4.39) shows explanatory information for the selected register.

Figure 4.39 Register View — Description Group



```
PTADD = 20

Data Direction Register for Port A

Bit Field Values:
   Pin 7 Data Directionbits[  7:7  ] = 0Input configuration
   Pin 6 Data Directionbits[  6:6  ] = 0Input configuration
   Pin 5 Data Directionbits[  5:5  ] = 1Output configuration
   Pin 4 Data Directionbits[  4:4  ] = 0Input configuration
   Pin 3 Data Directionbits[  3:3  ] = 0Input configuration
   Pin 2 Data Directionbits[  2:2  ] = 0Input configuration
   Pin 1 Data Directionbits[  1:1  ] = 0Input configuration
   Pin 0 Data Directionbits[  0:0  ] = 0Input configuration
```

The registers information covers:

- Name
- Current Value
- Description
- Bit field explanations and values

Some registers have multiple modes (meaning that the register’s bits can have multiple meanings, depending on the current mode). If the register you examine has multiple modes, you must select the appropriate mode.

Actions

Use the Actions group of the **Registers** view (Figure 4.40) to perform various operations on the selected register's bit-field values.

Figure 4.40 Register View — Actions Group

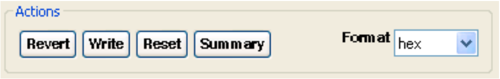


Table 4.31 lists each item in the **Actions** group and explains the purpose of each.

Table 4.31 Actions Group Items

Item	Description
Revert	Discard your changes to the current bit-field value and restore the last change you made. The debugger disables this button if you have not made any changes to the bit-field value.
Write	Save your changes to the current bit-field value and write those changes into the register's bit field. The debugger disables this button after writing the new bit-field value, or if you have not made any changes to that value.
Reset	Change each bit of the bit-field value to its register-reset value. The register takes on this value after a target-device reset occurs. To confirm the bit-field change, click **Write**. To cancel the change, click **Reset.**
Summary	Display **Description** group content in a pop-up window. Press the Esc key to close the pop-up window.
Format	Specify the data format of the displayed bit-field values.

Register Details Context Menu

To display the **Register Details** context menu, right-click on a bit-field value in the **Registers** view. This menu's commands duplicate capabilities available in the **Register Details** view.

Table 4.32 lists each command in the **Registers** view and explains the purpose of each.

Table 4.32 Register Details Context Menu

Menu Command	Description
Select All	Selects the entire contents of the current bit-field value
Copy Registers	Copies to selected bit fields content to the system clipboard
Enable	Lets the debugger to access the selected bit fields
Disable	Prevents the debugger from accessing the selected bit fields

Table 4.32 Register Details Context Menu (*continued*)

Menu Command	Description
Display as Array	Opens a dialog box that you can use to display the selected bit fields as an array of bit values
Cast to Type	Opens a dialog box that you can use to cast the selected bit field values to a different data type
Restore Original Type	Reverts the selected bit-field values to their default data types
Find	Opens a dialog box that you can use to select a particular register or bit field
Change Value	Opens a dialog box that you can use to change the current bit field value
Show Details As	Lets you specify how the debugger displays the register's contents. The options are: **Register Details Pane** — The register's values are displayed in a bit format, along with a description of their purpose. **Default Viewer** — The register's contents are displayed as a hexadecimal value
Add Register Group	Opens a dialog box that you can use to create a new collection of registers to display in the **Registers** view
Edit Register Group	Opens a dialog box that you can use to modify the collection of registers that the **Registers** view displays for the selected register group
Remove Register Group	Deletes the selected register group from the **Registers** view
Format	Use to specify the displayed data format of the selected bit field values: • Natural — Default data format • Binary — Binary data format • Decimal — Decimal data format • Hexadecimal — Hexadecimal data format
Create Watch Expression	Adds a new watch-expression entry to the Expressions view

Viewing Register Details

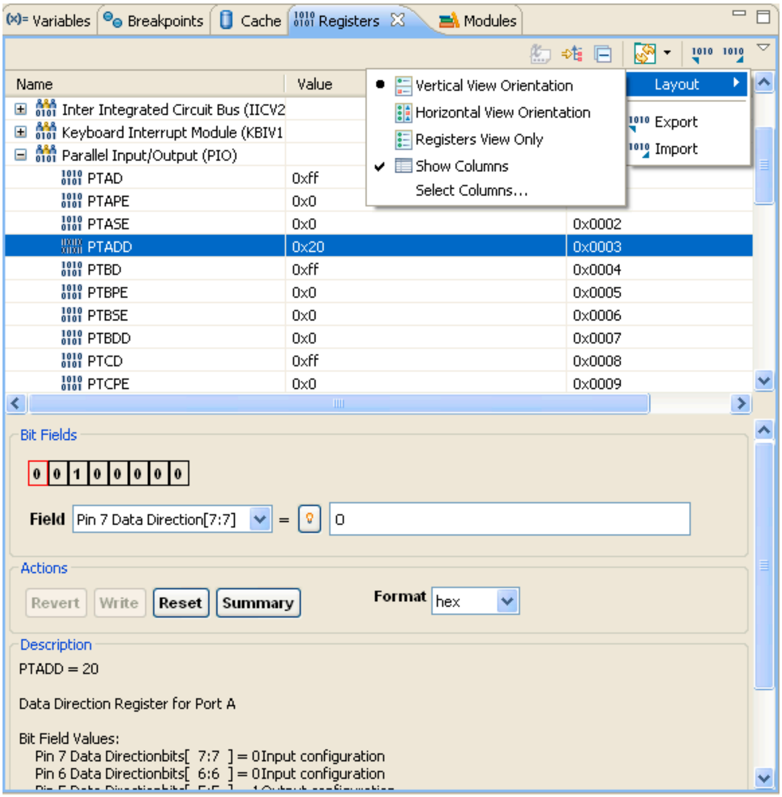
To open the **Registers** view, you must first start a debugging session.

To see the registers and their descriptions, perform these steps.

1. In the **Debug** perspective, click the **Registers** tab.

   The **Registers** view (Figure 4.41) appears.

Figure 4.41 Registers View, Register Details


(x)= Variables
Breakpoints
Cache
Registers
Modules
● Vertical View Orientation
Horizontal View Orientation
Registers View Only
✓ Show Columns
Select Columns...
Layout
Export
Import
Bit Fields
Actions
Revert
Write
Reset
Summary
Description

Name	Value	
⊞ Inter Integrated Circuit Bus (IICV2		
⊞ Keyboard Interrupt Module (KBIV1		
⊟ Parallel Input/Output (PIO)		
PTAD	0xff	
PTAPE	0x0	
PTASE	0x0	0x0002
PTADD	0x20	0x0003
PTBD	0xff	0x0004
PTBPE	0x0	0x0005
PTBSE	0x0	0x0006
PTBDD	0x0	0x0007
PTCD	0xff	0x0008
PTCPE	0x0	0x0009

0	0	1	0	0	0	0	0

Field Pin 7 Data Direction[7:7] = 0

Format hex

PTADD = 20

Data Direction Register for Port A

Bit Field Values:
Pin 7 Data Directionbits[7:7] = 0Input configuration
Pin 6 Data Directionbits[6:6] = 0Input configuration

2. Click the toolbar’s menu button (the inverted triangle highlighted in Figure 4.41).

3. Select **Layout > Vertical View Orientation** or **Layout > Horizontal View Orientation** to see the register details.

NOTE Selecting **Layout > Registers View Only** hides the register details.

4. Expand a register group to see individual registers.

5. Select a specific register by clicking on it.

   The debugger enables the appropriate buttons in the Actions group of the **Registers** view.

NOTE Use the **Format** list box to specify the format of data that appears in the **Registers** view.

6. Use the **Register** view to examine register details.

   For example, examine register details in these ways:

   - Use the **Bit Fields** group to see a graphical representation of the selected register's bit fields. You can use this graphical representation to select specific bits or bit fields.
   - Use the **Description** group to see an explanation of the selected register, bit field, or bit value.

TIP To enlarge the **Registers** view, click **Maximize** of the view's toolbar. After you finish looking at the register details, click **Restore** of the view's toolbar to return the view to its previous size. Alternatively, right-click the **Registers** tab and select **Detached**. The **Registers** view becomes a floating window that you can resize. After you finish looking at the register details, right-click the **Registers** tab of the floating window and select **Detached** again. You can rearrange the re-attached view by dragging its tab to a different collection of view tabs.

Setting Watchpoints

A watchpoint is another name for a data breakpoint. The debugger halts execution each time the watchpoint location is read, written, or accessed (read *or* written). The debugger lets you set a watchpoint on an address or range of addresses in memory.

NOTE The debugger does not support setting a watchpoint on a stack variable or a register variable. Watchpoint set on a local variable may result in halt of execution at unexpected locations.

You can set the watchpoint from the:

- **Add Watchpoint** dialog box
- **Breakpoints** view
- **Memory** view
- **Variables** view

Setting the watchpoint type defines the conditions under which the debugger halts execution.

The debugger handles both watchpoints and breakpoints in a similar way. You use the **Breakpoints** view to manage both types. For example, you use the **Breakpoints** view to add, remove, enable, and disable both watchpoints and breakpoints. The debugger attempts to set the watchpoint if a session is in progress based on the active debugging context (the active context is the selected project in the **Debug** view).

If the debugger sets the watchpoint when no debugging session is in progress, or when re-starting a debugging session, the debugger attempts to set the watchpoint at startup as it does for breakpoints. The **Problems** view displays error messages when the debugger fails to set a watchpoint. For example, if you set watchpoints on overlapping memory ranges, or if a watchpoint falls out of execution scope, an error message appears in the **Problems** view. You can use this view to see additional information about the error.

Add Watchpoint Dialog Box

Use the **Add Watchpoint** dialog box to create a watchpoint for a memory range. You can specify these parameters for a watchpoint:

- an address (including memory space)
- an expression that evaluates to an address
- a memory range
- an access type on which to trigger

To open the **Add Watchpoint** dialog box:

1. Open the **Debug** perspective.
2. Click one of these tabs:
    - Breakpoints
    - Memory
    - Variables

   The corresponding view appears.
3. Right-click the appropriate content inside the view as mentioned in Table 4.33.

Table 4.33 Opening Add Watchpoint Dialog Box

In the View...	Right-Click...
Breakpoints	an empty area inside the view.

Table 4.33 Opening Add Watchpoint Dialog Box (*continued*)

In the View...	Right-Click...
Memory	the cell or range of cells on which you want to set the watchpoint.
Variables	a global variable. Note that the debugger does not support setting a watchpoint on a stack variable or a register variable.

4. Select **Add Watchpoint (C/C++)** from the context menu.

   The **Add Watchpoint** dialog box appears as seen in Figure 4.42. The debugger sets the watchpoint according to the settings that you specify in the **Add Watchpoint** dialog box. The **Breakpoints** view shows information about the newly set watchpoint. The **Problems** view shows error messages when the debugger fails to set the watchpoint.

Figure 4.42 Add Watchpoint Dialog Box

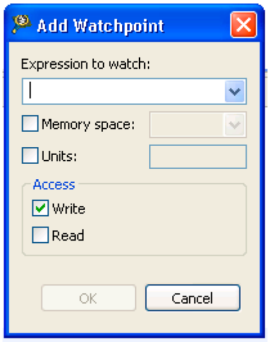


Table 4.34 describes the **Add Watchpoint** dialog box options.

Table 4.34 Add Watchpoint Dialog Box Options

Option	Description
Expression to watch	Enter an expression that evaluates to an address on the target device. The debugger displays an error message when the specified expression evaluates to an invalid address. You can enter these types of expressions: • An r-value, such as `&variable` • A register-based expression. Use the $ character to denote register names. For example, enter `$SP-12` to have the debugger set a watchpoint on the stack pointer address minus 12 bytes. The **Add Watchpoint** dialog box does not support entering expressions that evaluate to registers.
Memory space	Check this option to specify an address, including memory space, at which to set the watchpoint. Use the text box to specify the address or address range on which to set the watchpoint. If a debugging session is not active, the text/list box is empty, but you can still type an address or address range.
Units	Enter the number of addressable units that the watchpoint monitors.
Write	Check this option to enable the watchpoint to monitor write activity on the specified memory space and address range. Clear this option if you do not want the watchpoint to monitor write activity.
Read	Check this option to enable the watchpoint to monitor read activity on the specified memory space and address range. Clear this option if you do not want the watchpoint to monitor read activity.

Removing Watchpoints

To remove a watchpoint, perform these steps.

1. Open the **Breakpoints** view if it is not already open by choosing **Window > Show View > Breakpoints**.

   The **Breakpoint** view appears, displaying a list of watchpoints.

2. Right-click on the watchpoint you wish to remove and select **Remove** from the menu that appears.

   The selected watchpoint is removed, and it disappears from the list in the **Breakpoints** view.

Setting Breakpoints

The different breakpoint types that you can set are listed below:

- Software

  The debugger sets a software breakpoint into target memory. When program execution reaches the breakpoint, the processor stops and activates the debugger. The breakpoint remains in the target memory until you remove it.

  The breakpoint can only be set in writable memory like SRAM or DDR. You cannot use these type of breakpoints in ROM.

- Hardware

  Selecting the Hardware menu option causes the debugger to use the internal processor breakpoints. These breakpoints are usually very few and can be used with all types of memories (ROM/RAM) because they are implemented by using processor registers.

TIP You can also set breakpoint types by issuing the `bp` command in the CodeWarrior **Debugger Shell view.**

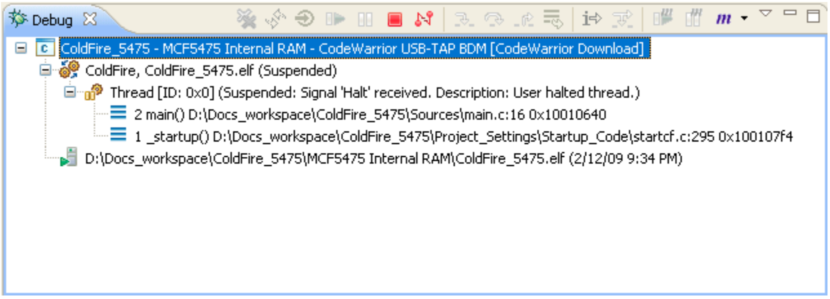
To set a breakpoint:

1. In the IDE's **Debug** perspective, click the **Debug** tab.

   Figure 4.43 shows the **Debug** view.

2. Right clicking on a code line will set a breakpoint.

Figure 4.43 Debug View



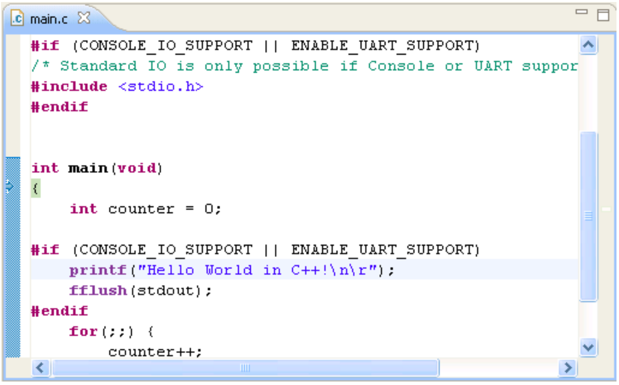
ColdFire_5475 - MCF5475 Internal RAM - CodeWarrior USB-TAP BDM [CodeWarrior Download]
ColdFire, ColdFire_5475.elf (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
2 main() D:\Docs_workspace\ColdFire_5475\Sources\main.c:16 0x10010640
1 _startup() D:\Docs_workspace\ColdFire_5475\Project_Settings\Startup_Code\startcf.c:295 0x100107f4
D:\Docs_workspace\ColdFire_5475\MCF5475 Internal RAM\ColdFire_5475.elf (2/12/09 9:34 PM)

3. Expand the **Thread** group.

4. Under the **Thread** group, select the thread that has the `main()` function.

   The source code appears in an editor view as in Figure 4.44. The small blue arrow to the left of the source code indicates which code statement the processor's program counter is set to execute next.

Figure 4.44 Editor View



```
#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
/* Standard IO is only possible if Console or UART suppor
#include <stdio.h>
#endif


int main(void)
{
    int counter = 0;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++!\n\r");
    fflush(stdout);
#endif
    for(;;) {
        counter++;
```

5. In the editor view, place the cursor on the line that has this statement:

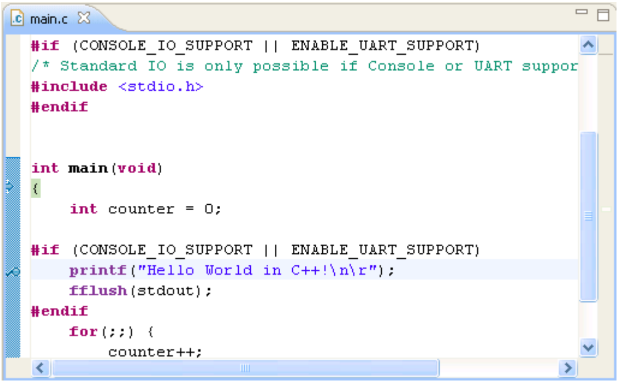
   `printf(“Hello World in C++!\n\r”);`

6. Select **Run** > **Toggle Line Breakpoint**.

7. A blue dot appears in the marker bar to the left of the line as seen in Figure 4.45. This dot indicates an enabled breakpoint. After the debugger installs the breakpoint, a blue checkmark appears beside the dot. The debugger installs a breakpoint by loading into the Java™ virtual machine the code in which you set that breakpoint.

TIP An alternate way to set a breakpoint is to double-click the marker bar to the left of any source-code line. If you set the breakpoint on a line that does not have an executable statement, the debugger moves the breakpoint to the closest subsequent line that has an executable statement. The marker bar shows the installed breakpoint location. If you want to set a hardware breakpoint instead of a software breakpoint, use the `bp` command in the **Debugger Shell** view. You can also right-click on the marker bar to the left of any source-code line, and select **Set Special Breakpoint** from the context menu.

Figure 4.45 Editor View — After Setting Breakpoints

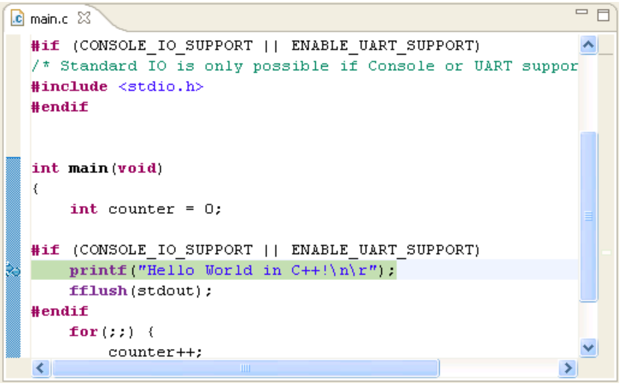


```
#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
/* Standard IO is only possible if Console or UART suppor
#include <stdio.h>
#endif


int main(void)
{
    int counter = 0;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++!\n\r");
    fflush(stdout);
#endif
    for(;;) {
        counter++;
```

8. From the menu bar, select **Run** > **Resume**.

   The debugger executes all lines up to, but not including, the line at which you set the breakpoint. The editor view highlights the line at which the debugger suspended execution as shown in Figure 4.46. Note that the program counter (blue arrow) is positioned here.

Figure 4.46 Editor View — After Reaching Breakpoint



```
#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
/* Standard IO is only possible if Console or UART suppor
#include <stdio.h>
#endif


int main(void)
{
    int counter = 0;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++!\n\r");
    fflush(stdout);
#endif
    for(;;) {
        counter++;
```

Setting Hardware Breakpoints

There are two ways to set hardware breakpoints. These are:

- Using IDE to Set Hardware Breakpoint
- Using Debugger Shell to Set Hardware Breakpoint

Using IDE to Set Hardware Breakpoint

In either the **C/C++** perspective or the **Debug** perspective, select the source line in the **Editor** view where you want to place the breakpoint. Go to the marker bar on the left side of the **Editor** view. Right-click on it to display a menu. Select **Set Special Breakpoint > Hardware** to set a hardware breakpoint.

Using Debugger Shell to Set Hardware Breakpoint

Use the **Debugger Shell** view to set hardware breakpoints. Perform these steps to use the debugger shell to set a hardware breakpoint:

1. Open the **Debugger Shell** view.
2. Begin the command line with the text: `bp -hw`
3. Complete the command line by specifying the function, address, or file at which you want to set the hardware breakpoint.

   For example, to set a breakpoint at line 6 in the source file `main.c`, type:

   ```
   bp -hw main.c 6
   ```
4. Press the Enter key.

   The debugger shell executes the command and sets the hardware breakpoint.

TIP Enter `help bp` at the command-line prompt to see examples of the `bp` command syntax and usage.

Removing Breakpoints

To remove a breakpoint from your program, you have two options. These are:

- Remove Breakpoint Using Marker Bar
- Remove Breakpoint Using Breakpoints View

NOTE For more information on removing hardware breakpoints, refer to the topic Removing Hardware Breakpoints.

Remove Breakpoint Using Marker Bar

To remove an existing breakpoint using the marker bar, perform these steps.

1. Right-click on the existing breakpoint in the marker bar.
2. Select **Toggle Breakpoint** from the menu that appears.

Remove Breakpoint Using Breakpoints View

To remove an existing breakpoint using the **Breakpoints** view, perform these steps.

1. Open the **Breakpoints** view if it is not already open by choosing **Window > Show View > Breakpoints**.

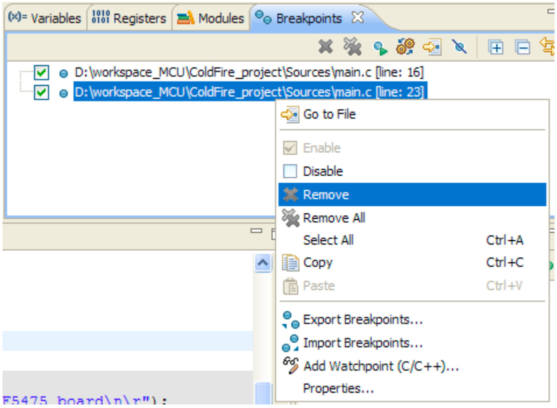
   The **Breakpoint** view appears, displaying a list of breakpoints.

2. Right-click on the breakpoint you want to remove and select **Remove** from the context menu (Figure 4.47).

   The selected breakpoint is removed, and it disappears from the both the marker bar and the list in the view.

NOTE To remove all of the breakpoints from the program at once, select **Remove All** from the context menu.

Figure 4.47 Removing Breakpoint


Variables
Registers
Modules
Breakpoints
Go to File
Enable
Disable
Remove
Remove All
Select All Ctrl+A
Copy Ctrl+C
Paste Ctrl+V
Export Breakpoints...
Import Breakpoints...
Add Watchpoint (C/C++)...
Properties...
F5475 board\n\r");

☑ D:\workspace_MCU\ColdFire_project\Sources\main.c [line: 16]
☑ D:\workspace_MCU\ColdFire_project\Sources\main.c [line: 23]

Removing Hardware Breakpoints

There are two ways to remove existing hardware breakpoints. These are:

- Remove Hardware Breakpoint Using IDE
- Remove Hardware Breakpoint Using Debugger Shell

Remove Hardware Breakpoint Using IDE

To remove a hardware breakpoint, perform these steps.

1. Right-click on the existing breakpoint in the marker bar.
2. Select **Toggle Breakpoint** from the menu that appears.

Alternatively, you may remove the breakpoint from the Breakpoint view by performing these steps.

1. Open the **Breakpoints** view if it is not already open by choosing **Window > Show View > Breakpoints**.

   The **Breakpoint** view appears, displaying a list of breakpoints.

2. Right-click on the hardware breakpoint you wish to remove and select **Remove** from the menu that appears, as shown in Figure 4.47.

   The selected breakpoint is removed, and it disappears from the both the marker bar and the list in the view.

Remove Hardware Breakpoint Using Debugger Shell

To remove a hardware breakpoint using the **Debugger Shell** view, perform these steps.

1. Open the **Debugger Shell** view.
2. Begin the command line with the text: `bp#`
3. Complete the command line by specifying the function, address, or file at which you want to remove the hardware breakpoint.

   For example, to remove a breakpoint at line 6 in the source file `main.c`, type:

   ```
   bp#<bp_index> off
   ```

   Where, `<bp_index>` is the index if the hardware breakpoint is at line 6.

4. Press the Enter key.

   The debugger shell executes the command and removes the hardware breakpoint.

Setting Stack Crawl Depth

Select the **Maximum stack crawl depth** command lets you set the depth of the stack to read and display. Showing all levels of calls when you are examining function calls several levels deep can sometimes make stepping through code more time-consuming. Therefore, you can use this menu option to reduce the depth of calls that the debugger displays.

To set the stack crawl depth, perform these steps.

1. Select **Window** > **Preferences**.

   The **Preferences** dialog box appears.

2. Expand the **C/C++** tree control and select **Debug**.

   The general settings for C/C++ debugging are displayed on the right-hand side of the **Preferences** dialog box.

3. Specify the appropriate stack crawl depth, in the **Maximum stack crawl depth** text box.

NOTE You can specify any integer from 1 to 100.

Changing Program Counter Value

To change the program-counter value, perform these steps.

1. Start a debugging session.
2. In the **Editor** view, place the cursor on the line that you want the debugger to execute next.
3. Right-click in the **Editor** view.

   A context menu appears.

4. From the context menu, select **Move To Line**.

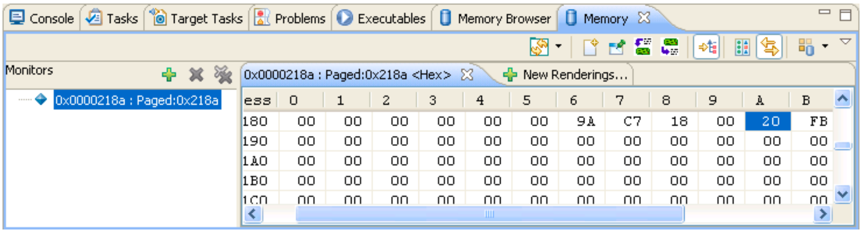
   CodeWarrior IDE modifies the program counter to the specified location. The **Editor** view shows the new location.

Viewing Memory

Use the **Memory** view to examine the active memory rendering of a specified expression or address. To display this view from the **Debug** perspective, Select **Window > Show View > Memory**, and the **Memory** view appears.

The **Memory** view supports the display of multiple memory spaces. Figure 4.48 shows the **Memory** view with the *Expression:baseaddr <name> tree* active memory rendering tab.

Figure 4.48 Memory View



ess	0	1	2	3	4	5	6	7	8	9	A	B
180	00	00	00	00	00	00	9A	C7	18	00	20	FB
190	00	00	00	00	00	00	00	00	00	00	00	00
1A0	00	00	00	00	00	00	00	00	00	00	00	00
1B0	00	00	00	00	00	00	00	00	00	00	00	00
1C0	00	00	00	00	00	00	00	00	00	00	00	00

Adding Memory Monitor

You can add multiple memory monitors to the **Memory** view. To add a new memory monitor, perform these steps.

1. Start a debugging session.
2. Open the **Memory** view.
3. Click the plus-sign ✚ icon on the **Monitors** pane toolbar. Alternatively, right-click in the **Monitors** pane and select **Add Memory Monitor** from the context menu.
4. The **Monitor Memory** dialog box (Figure 4.49) appears.

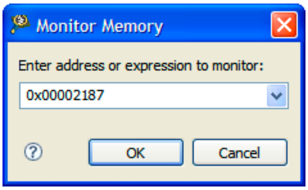
NOTE The **Enter memory space and address** option appears only when the debugger associated with the active debugging context supports memory spaces, and the currently debugged process has multiple memory spaces.

5. Specify options as explained in Table 4.35.

Table 4.35 Monitor Memory Dialog Box Options

Option	Description
Enter address or expression	Enter the expression to monitor in decimal or hexadecimal values. You can use the drop-down list to select a previously specified expression.
Memory Space	Check to specify the memory space (*Paged or Flash*).

Figure 4.49 Monitor Memory Dialog Box



Enter address or expression to monitor:
0x00002187

6. Click **OK**.

   The memory monitor appears in the **Memory** view (Figure 4.48).

Adding Memory Rendering

You can use the **Renderings** pane of the **Memory** view to examine the memory content, starting at any valid address. The information displayed in this page is read only and cannot be used to modify the memory content.

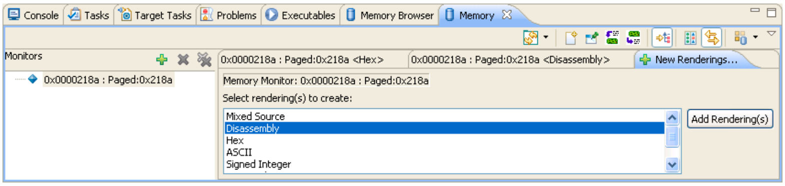
To add a new memory rendering, perform these steps.

1. Start a debugging session.
2. Open the **Memory** view.
3. In the **Monitors** pane, select the memory monitor for which you want to add a memory rendering.

NOTE To create a memory monitor, right-click a blank area in the **Monitors** pane and select **Add Memory Monitor**. Alternatively, click the plus-sign ✚ icon in the **Monitors** pane toolbar.

4. Click the **New Renderings** tab to select renderings.

Figure 4.50 New Renderings View


Console
Tasks
Target Tasks
Problems
Executables
Memory Browser
Memory
Monitors
0x0000218a : Paged:0x218a
0x0000218a : Paged:0x218a <Hex>
0x0000218a : Paged:0x218a <Disassembly>
New Renderings...
Add Rendering(s)

Memory Monitor: 0x0000218a : Paged:0x218a

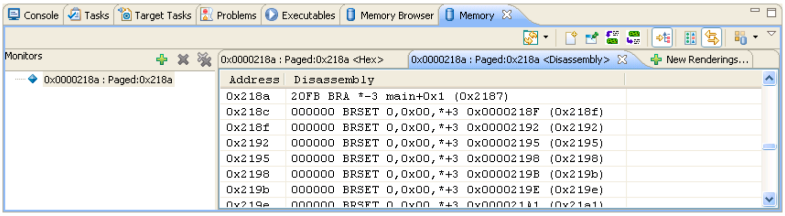
Select rendering(s) to create:

Mixed Source
Disassembly
Hex
ASCII
Signed Integer

5. Select a rendering type from the **Select rendering(s) to create** list and click the **Add Rendering(s)** button. Alternatively, right-click in the **Renderings** pane and select **Add Rendering** from the context menu. For example, select *Disassembly*.
6. Click **Rendering(s)**.

   The selected memory rendering type appears in the **Memory** view.

Figure 4.51 Added Rendering


Console
Tasks
Target Tasks
Problems
Executables
Memory Browser
Memory
Monitors
0x0000218a : Paged:0x218a
0x0000218a : Paged:0x218a <Hex>
0x0000218a : Paged:0x218a <Disassembly>
New Renderings...

```
Address   Disassembly
0x218a    20FB BRA *-3 main+0x1 (0x2187)
0x218c    000000 BRSET 0,0x00,*+3 0x0000218F (0x218f)
0x218f    000000 BRSET 0,0x00,*+3 0x00002192 (0x2192)
0x2192    000000 BRSET 0,0x00,*+3 0x00002195 (0x2195)
0x2195    000000 BRSET 0,0x00,*+3 0x00002198 (0x2198)
0x2198    000000 BRSET 0,0x00,*+3 0x0000219B (0x219b)
0x219b    000000 BRSET 0,0x00,*+3 0x0000219E (0x219e)
0x219e    000000 BRSET 0,0x00,*+3 0x000021A1 (0x21a1)
```

Removing Memory Rendering

To remove a memory rendering from the **Memory** view, perform these steps.

1. Open the **Memory** view.
2. In the **Renderings** pane, select the tab that corresponds to the memory rendering that you want to remove.
3. Click the cross-sign ✖ icon on the **Renderings** pane toolbar. Alternatively, right-click on the **Renderings** pane and select **Remove Rendering** from the context menu.

   The memory rendering is removed from the **Memory** view.

Resetting to Base Address

To reset the memory rendering and display the base address of the rendering, perform these steps.

1. Open the **Memory** view.
2. In the **Renderings** pane, select the tab that corresponds to the disassembly rendering that you want to reset to the base address.
3. Right-click in the **Renderings** pane and select **Reset to Base** from the context menu.
4. The disassembly rendering scrolls to the line that contains the base address of the displayed rendering.

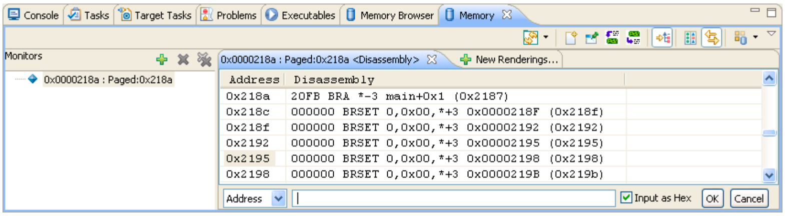
Go to Address

The memory view provides graphical controls to display memory at a specific address. To go to a specific address, perform these steps.

1. Open the **Memory** view.
2. In the **Renderings** pane, select the tab that corresponds to the disassembly rendering for which you want to display a specific address.
3. Right-click in the **Renderings** pane and select **Go to Address** from the context menu.

   A group of controls appears on the **Renderings** pane (Figure 4.52).
4. In the blank text box, enter the address that you want to display.

Figure 4.52 Disassembly Rendering - Go to Address



Address	Disassembly
0x218a	20FB BRA *-3 main+0x1 (0x2187)
0x218c	000000 BRSET 0,0x00,*+3 0x0000218F (0x218f)
0x218f	000000 BRSET 0,0x00,*+3 0x00002192 (0x2192)
0x2192	000000 BRSET 0,0x00,*+3 0x00002195 (0x2195)
0x2195	000000 BRSET 0,0x00,*+3 0x00002198 (0x2198)
0x2198	000000 BRSET 0,0x00,*+3 0x0000219B (0x219b)

NOTE Check the **Input as Hex** checkbox only if you enter the address in hexadecimal notation.

5. Click **OK** to have the Disassembly rendering scroll to the specified address. Alternatively, click **Cancel** to abort the operation and hide the group of controls.

Hard Resetting

Use the `reset hard` command in the **Debugger Shell** view to send a hard reset signal to the target processor.

NOTE The **Hard Reset** command is enabled only if the debug hardware you are using supports it.

Debugging Externally Built Executable Files

You can use the Microcontrollers ELF and PA ELF executable wizards to debug .elf files generated by a different IDE.

The main purpose of the **MCU Executable Import** and **PA ELF Executable** wizards is to create a CodeWarrior for Microcontrollers Eclipse project that can be readily debugged starting from an executable file build with a Microcontrollers toolchain.

The **MCU Executable Import** and **PA ELF Executable** wizards let you import a `*.elf`, `*.abs`, or `*.flt` file and associate it to a project.

This topic describes the various pages of the two wizards and the list steps that assists you in importing and associating an executable (`*.elf`, `*.abs`, or `*.flt`) file to a project.

- Microcontrollers ELF Executable
- Debug an Externally Built Microcontrollers Executable File

Microcontrollers ELF Executable

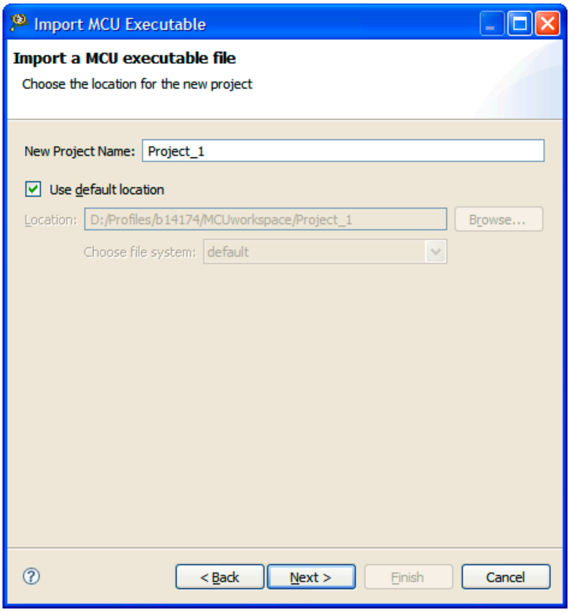
The pages in the Microcontrollers ELF Executable are:

- Import a MCU Executable File Page
- Select MCU executable file to be imported Page
- Device and Board Page
- Connections Page

Import a MCU Executable File Page

Use this page to name your new project, and specify the workspace directory.

Figure 4.53 Import MCU Executable — Import a MCU Executable File Page



Import MCU Executable
Import a MCU executable file
Choose the location for the new project

New Project Name: Project_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/Project_1

Choose file system: default

Browse...
< Back
Next >
Finish
Cancel

Table 4.36 describes the purpose of the various options.

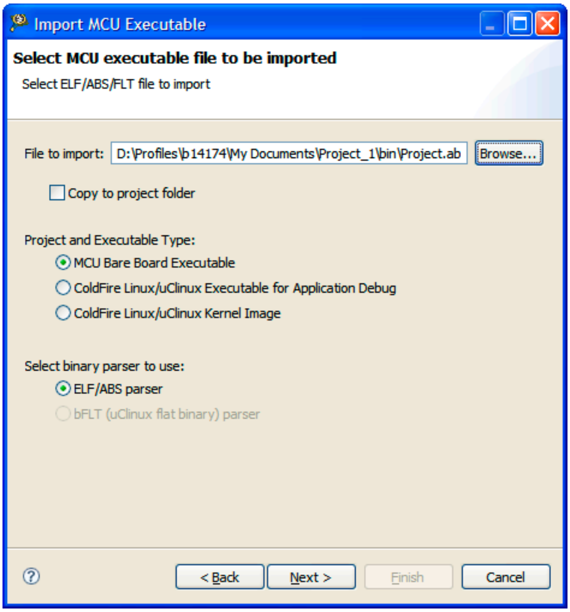
Table 4.36 Import a MCU Executable File Page Settings

Option	Description
New Project Name	Enter the name for the new project in this text box.
Use default location	Stores the files required to build the program in the Workbench's current workspace directory. The project files are located in the directory you specify. Use the **Location** option to select the directory.
Location	Specifies the directory that contains the project files. Click **Browse** to navigate to the desired directory. This option is only available when **Use default location** is clear.
Choose file system	Specifies the file system to use. However, this option is available only when **Use default location** is clear. You can select either the default file system or a Remote System Explorer (RSE).

Select MCU executable file to be imported Page

Use this page to select an existing Microcontrollers `ELF, ABS, or FLT` file you want to import.

Figure 4.54 Import MCU Executable — Select MCU executable file to be imported Page



Import MCU Executable
Select MCU executable file to be imported
Select ELF/ABS/FLT file to import

File to import: D:\Profiles\b14174\My Documents\Project_1\bin\Project.ab

Browse...

☐ Copy to project folder

Project and Executable Type:

- ● MCU Bare Board Executable
- ○ ColdFire Linux/uClinux Executable for Application Debug
- ○ ColdFire Linux/uClinux Kernel Image

Select binary parser to use:

- ● ELF/ABS parser
- ○ bFLT (uClinux flat binary) parser

< Back
Next >
Finish
Cancel

Table 4.37 explains the options available on the page.

Table 4.37 Select MCU executable file to be imported Page Settings

Option	Description
File to Import	Specifies the path of the ELF, ABS, or FLT files.
Browse	Click to locate the directory that contains the `*.elf`, `*.abs`, or `*.flt` files.
Copy to project	Check to copy the selected import file in the new project.
MCU Bare Board Executable	Select to use the microcontrollers bareboard executable.
ColdFire Linux/uClinux Executable for Application Debug	Select to use the ColdFire Linux/uClinux executable for application debug.

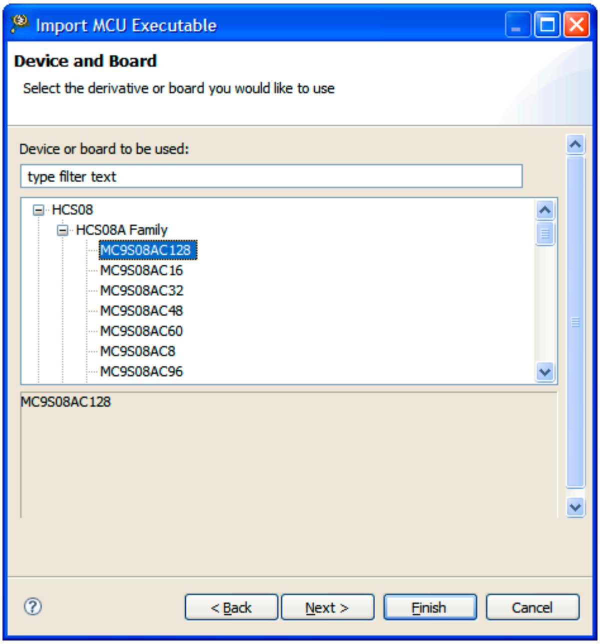
Table 4.37 Select MCU executable file to be imported Page Settings

Option	Description
ColdFire Linux/uClinux Kernel Image	Select to use the ColdFire Linux/uClinux Kernel Image.
Select binary parser	Select a binary parser for the executable to be imported into the CodeWarrior IDE. The drop-down list includes various parsers supported by the CodeWarrior IDE.The commonly used parsers are: • ELF/ABS parser • bFLT (uClinux flat binary) parser

Device and Board Page

Use this page to select the derivative or board you would like to use.

Figure 4.55 Import MCU Executable — Devices Page

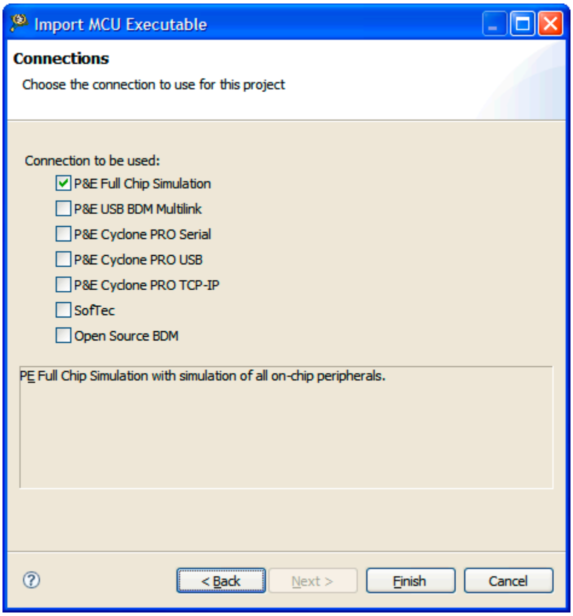


Import MCU Executable
Device and Board
Select the derivative or board you would like to use
Device or board to be used:
type filter text
HCS08
HCS08A Family
MC9S08AC128
MC9S08AC16
MC9S08AC32
MC9S08AC48
MC9S08AC60
MC9S08AC8
MC9S08AC96
MC9S08AC128
< Back
Next >
Finish
Cancel

Connections Page

Use this page to select a connection to use for the project. Depending on the selected derivative or board, the connections will appear enabled or grayed out.

Figure 4.56 Import MCU Executable — Connections Page



Connections

Choose the connection to use for this project

Connection to be used:

- ☑ P&E Full Chip Simulation
- ☐ P&E USB BDM Multilink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO TCP-IP
- ☐ SofTec
- ☐ Open Source BDM

PE Full Chip Simulation with simulation of all on-chip peripherals.

Debug an Externally Built Microcontrollers Executable File

You can use the Microcontrollers ELF Executable wizard to debug an .elf file generated by a different IDE.

To debug externally built executable files, perform these steps.

1. Import a MCU Executable File Page
2. Specify Executable File to Import
3. Select Derivative or Board
4. Select Connection
5. Edit Launch Configuration
6. Specify Source Lookup Path

7. Debug Executable File

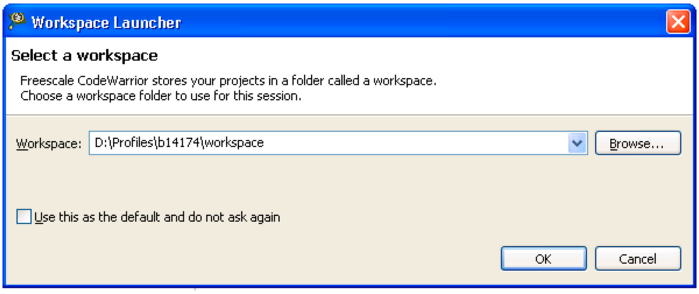
Import a MCU Executable File

You specify the externally built executable file that you want to debug in the CodeWarrior IDE. The IDE imports the executable file into a new project. To specify the executable file, perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 4.57 WorkSpace Launcher Dialog Box



Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Profiles\b14174\workspace

☐ Use this as the default and do not ask again

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

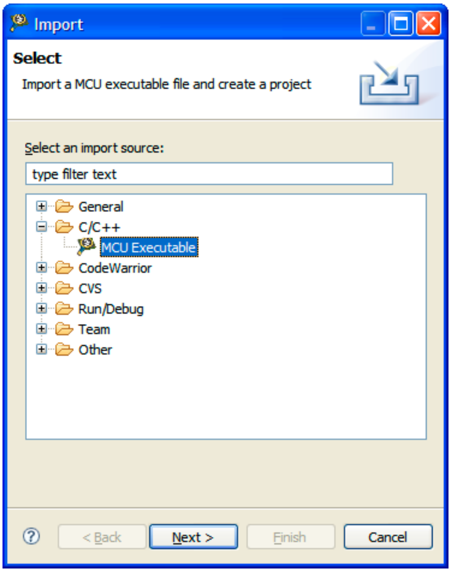
4. Select **File > Import,** from the IDE menu bar.

   The **Import** wizard appears.

5. Expand the **C/C++** group.

6. Select **MCU Executable** to debug a Microcontrollers `*.elf, *.abs,` or `*.flt` file as shown in Figure 4.58.

Figure 4.58 Import Wizard — Select MCU Executable


Import
Select
Import a MCU executable file and create a project
Select an import source:
type filter text
General
C/C++
MCU Executable
CodeWarrior
CVS
Run/Debug
Team
Other
< Back
Next >
Finish
Cancel

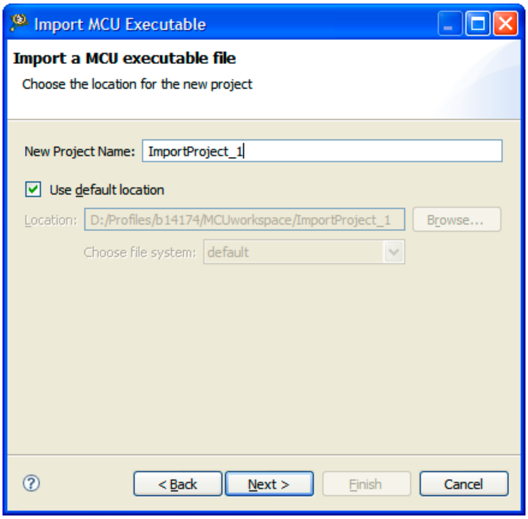
7. Click **Next**.

   The **Import a MCU executable file** page (Figure 4.59) appears.

8. Specify a name for the new project. For example, enter the project name as `ImportProject_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 4.59 Import MCU Executable — Import a MCU executable file Page


Import MCU Executable
Import a MCU executable file
Choose the location for the new project
Browse...
Choose file system:
< Back
Next >
Finish
Cancel

New Project Name: ImportProject_1

☑ Use default location

Location: D:/Profiles/b14174/MCUworkspace/ImportProject_1

Choose file system: default

9. Click **Next**.

   The **Select MCU executable file to be imported** page appears.

Specify Executable File to Import

1. Click **Browse**.

   The **Select file** dialog box appears.

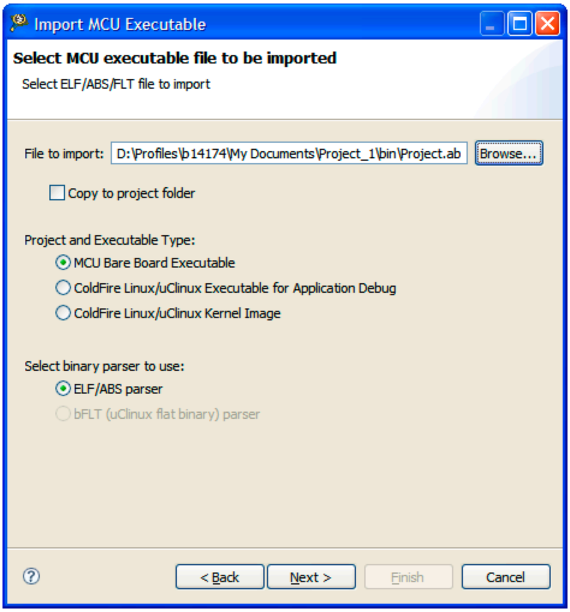
2. Navigate to the executable file that you want to import and click **Open**.

   The path of the selected file appears in the **File to import** text box.

3. Check the **Copy to project** checkbox if you want to copy the specified file in the new project. By default, the **Copy to project** checkbox is cleared.

4. From the **Select binary parser to use**, select the parser you want to use. (Figure 4.60).

Figure 4.60 Import MCU Executable — Specify MCU Executable File to Import



Import MCU Executable

Select MCU executable file to be imported

Select ELF/ABS/FLT file to import

File to import: D:\Profiles\b14174\My Documents\Project_1\bin\Project.ab

☐ Copy to project folder

Project and Executable Type:
- ● MCU Bare Board Executable
- ○ ColdFire Linux/uClinux Executable for Application Debug
- ○ ColdFire Linux/uClinux Kernel Image

Select binary parser to use:
- ● ELF/ABS parser
- ○ bFLT (uClinux flat binary) parser

5. Click **Next**.

   The **Devices** page appears.

Select Derivative or Board

1. Expand the tree control and select the derivative or board you would like to use. For example, select `HCS08 > HCS08A Family > MC9S08AC128`.
2. Click **Next**.

   The **Connections** page appears.

Select Connection

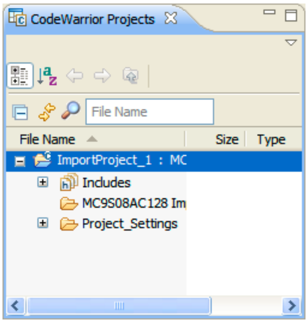
1. Select the desired connection from the **Connection to be used** group. For example, check the **P&E Full Chip Simulation** checkbox.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

2. Click **Finish**.

   The **Import MCU Executable** window closes and the wizard creates a project according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Figure 4.61 CodeWarrior Projects View



3. Right-click on the project and from the context menu select **Build Project**.

   The new project is ready for use. You can now customize it by adding your own source code files, changing debugger settings, or adding libraries.

Edit Launch Configuration

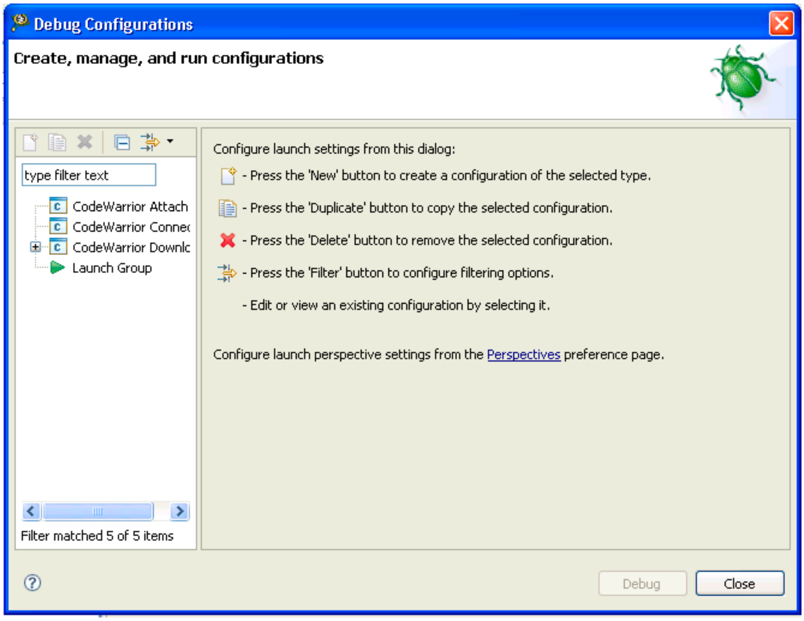
Before you edit the launch configuration, ensure that you create a project for the executable file.

To edit the launch configuration for your executable file, perform these steps.

1. From the main menu bar of the IDE, select **Run > Debug Configurations**. The IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

   The **Debug Configurations** dialog box appears. The left side of this dialog box has a list of debug configurations that apply to the current application.

Figure 4.62 Debug Configurations Dialog Box



Debug Configurations
Create, manage, and run configurations
type filter text
CodeWarrior Attach
CodeWarrior Connec
CodeWarrior Downlc
Launch Group
Filter matched 5 of 5 items
Debug
Close

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.

- Press the 'Delete' button to remove the selected configuration.

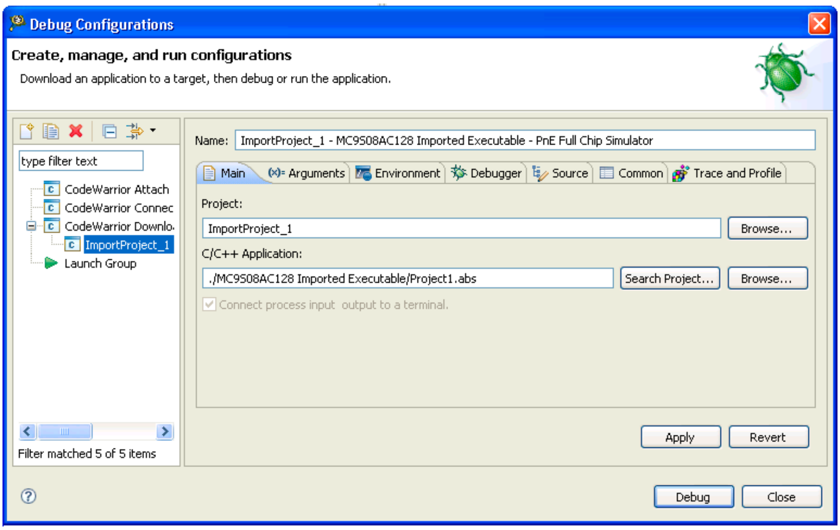
- Press the 'Filter' button to configure filtering options.

- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

2. Expand the **CodeWarrior Download** configuration.
3. From the expanded list, select the newly created debug configuration. For example, select *ImportProject_1 - MC9S08AC128*.

Figure 4.63 Selected Launch Configuration



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: ImportProject_1 - MC9S08AC128 Imported Executable - PnE Full Chip Simulator

Project: ImportProject_1

C/C++ Application: ./MC9S08AC128 Imported Executable/Project1.abs

☑ Connect process input output to a terminal.

type filter text
CodeWarrior Attach
CodeWarrior Connec
CodeWarrior Downlo
ImportProject_1
Launch Group
Filter matched 5 of 5 items
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Browse...
Search Project...
Browse...
Apply
Revert
Debug
Close

4. Click the **Debugger** tab of the **Debug Configurations** dialog box.

   The corresponding page appears.

5. Use the **Debugger** list box to specify the debugger that corresponds to your type of executable file.

6. Configure the debugger options as appropriate for your executable file.

   For example, specify the appropriate target processor, any initialization files, and connection protocol.

Specify Source Lookup Path

You need to specify the source lookup path in terms of the compilation path and the local file-system path for the newly imported executable file. The CodeWarrior debugger uses both of these paths to debug the executable file.

The compilation path is the path to the original project that built the executable file. If the original project is from an IDE on a different computer, you specify the compilation path in terms of the file system on that computer.

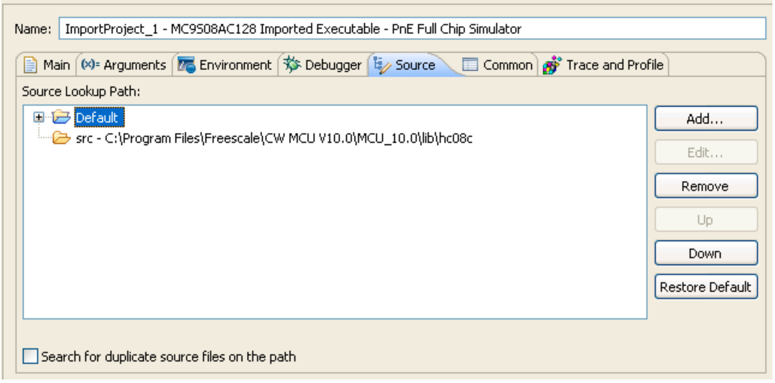
The local file-system path is the path to the project that the CodeWarrior IDE creates in order to debug the executable file.

To specify a source lookup path for your executable file:

1. Click the **Source** tab of the **Debug Configurations** dialog box.

   The corresponding page (Figure 4.64) appears.

Figure 4.64 Debug Configurations Dialog Box — Source Page


Name: ImportProject_1 - MC9S08AC128 Imported Executable - PnE Full Chip Simulator

Source Lookup Path:

Default
src - C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\lib\hc08c

☐ Search for duplicate source files on the path

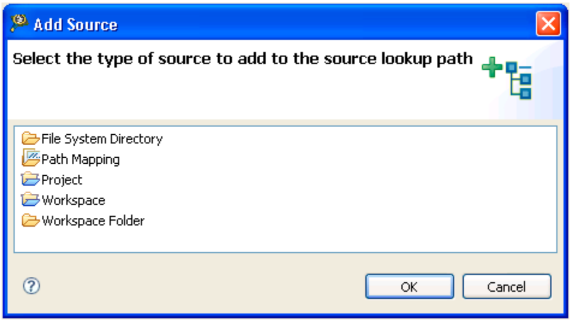
Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Add...
Edit...
Remove
Up
Down
Restore Default

2. Click **Add**.

   The **Add Source** dialog box appears.

3. Select **Path Mapping** (Figure 4.65).

Figure 4.65 Add Source Dialog Box


Add Source

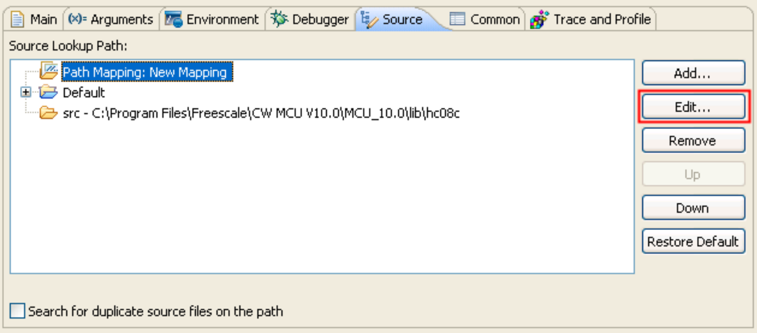
Select the type of source to add to the source lookup path

File System Directory
Path Mapping
Project
Workspace
Workspace Folder

4. Click **OK**.

   The **Add Source** dialog box closes. The IDE selects the new mapping in the **Source Lookup Path** list of the **Source** page.

Figure 4.66 Source Lookup Path



Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Add...
Edit...
Remove
Up
Down
Restore Default

Source Lookup Path:

- Path Mapping: New Mapping
- Default
- src - C:\Program Files\Freescale\CW MCU V10.0\MCU_10.0\lib\hc08c

☐ Search for duplicate source files on the path

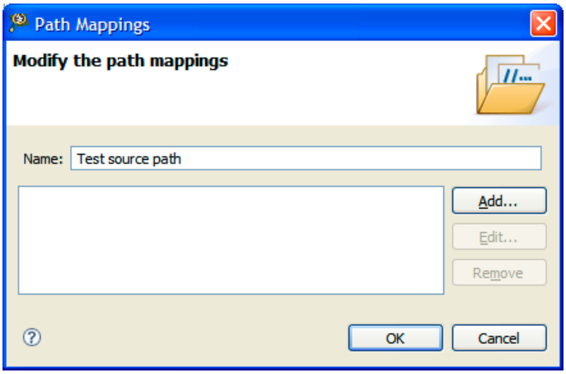
5. Click the **Edit** button of the **Source** page.

   The **Path Mappings** dialog box (Figure 4.67) appears.

6. In the **Name** text box, enter the name of the new path mapping.

   The name you enter also appears in the **Source Lookup Path** list of the **Source** page.

Figure 4.67 Path Mappings Dialog Box



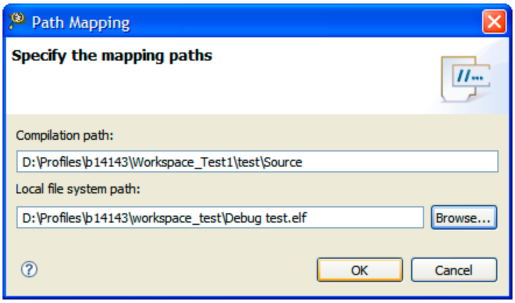
Path Mappings
Modify the path mappings
Add...
Edit...
Remove
OK
Cancel

Name: Test source path

7. Click **Add**.

   The **Path Mapping** dialog box (Figure 4.68) appears.

Figure 4.68 Path Mapping Dialog Box



Compilation path: D:\Profiles\b14143\Workspace_Test1\test\Source

Local file system path: D:\Profiles\b14143\workspace_test\Debug test.elf

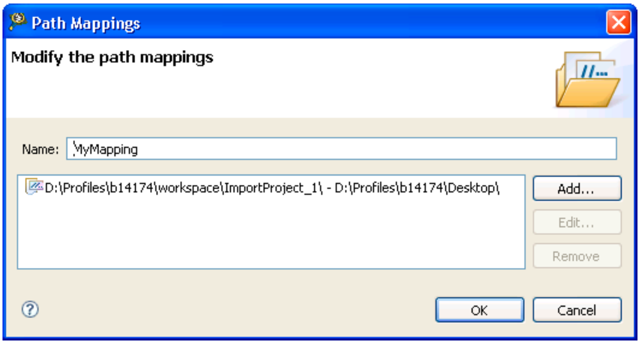
8. In the **Compilation path** text box, enter the path to the parent project of the executable file, relative to the computer that generated the file.

TIP You can use the IDE to discover the path to the parent project of the executable file, relative to the computer that generated the file. In the **C/C++ Projects** view of the **C/C++** perspective, expand the project that contains the executable file that you want to debug. Next, expand the group that has the name of the executable file itself. A list of paths appears, relative to the computer that generated the file. Search this list for the names of source files used to build the executable file. The path to the parent project of one of these source files is the path you should enter in the **Compilation path** text box.

9. In the **Local file system path** text box, enter the path to the parent project of the executable file, relative to your computer. Alternatively, click the **Browse** button to specify the parent project.
10. Click **OK**.

    The **Path Mapping** dialog box closes. The mapping information now appears in the **Path Mappings** dialog box.

Figure 4.69 Path Mappings Dialog Box



Name: MyMapping

D:\Profiles\b14174\workspace\ImportProject_1\ - D:\Profiles\b14174\Desktop\

11. Click **OK**.

    The **Path Mappings** dialog box closes. The mapping information now appears under the path mapping shown in the **Source Lookup Path** list of the **Source** page.

12. If required, change the order in which the IDE searches the paths.

    The IDE searches the paths in the order shown in the **Source Lookup Path** list, stopping at the first match. To change this order, select a path, then click the **Up** or **Down** button to change its position in the list.

13. Click **Apply**.

    The IDE saves your changes.

Debug Executable File

Use the CodeWarrior debugger to debug the externally built executable file.

To debug the executable file, click the **Debug** button of the **Debug Configurations** dialog box.

5

Multicore Debugging

This chapter lists the steps to define multiple, arbitrary groupings of cores and perform multicore operations. This is of particular importance in the case that the hardware has provided a means to synchronize an operation across multiple cores.

Additionally, the chapter lists the steps to add multicore operations to Eclipse through both the UI and through the Debugger Shell. The operations are divided into two sets. One set is for controlling target execution state, also known as run control, and consists of Resume, Suspend and Step Into. The other set is for controlling debug session lifetime, which we'll refer to as session control, and consists of Restart and Terminate. An integral part of the feature will be the ability to select any subset of cores for operation.

The topic in this chapter is:

- Creating LSM/DPM Projects
- Debugging LSM/DPM Projects
- Debugging Multicore Projects
- Editing Multicore Groups
- Editing System Types

Creating LSM/DPM Projects

The Power Architecture e200 Lock-Step Mode (LSM) and Dual Processor Mode (DPM) projects are ideal examples of a multicore project.

- Creating LSM Project
- Creating DPM Project

MPC5643L devices can operate in two modes of operation:

• Lock-Step Mode (LSM) — This mode takes its name from the execution of the same commands by both cores in synchronicity (lock-step).

• Decoupled Parallel Mode (DPM) — In this mode, each CPU core and connected channel run independently from the other one and redundancy checkers (RCCU) are disabled. MPC5643L devices support only static configuration at power-on (either LS or DP modes).

NOTE One of the two modes is statically selected at power-up. The selected mode may be changed only going through a full power-on reset. Each of these modes has several specific submodes that the device can enter. These modes differ, for example, in the list of enabled modules, pin configurations, reset phase, and safety status.

Lock-Step Mode (LSM) is used to increase safety, Dual Processor Mode (DPM) is used to increase performance. 1 Device is in Lockstep Mode (LSM) and 0 Device is in Dual Processor Mode (DPM).

Creating LSM Project

To create a LSM, perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `Project_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

6. Click **Next**.

   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.

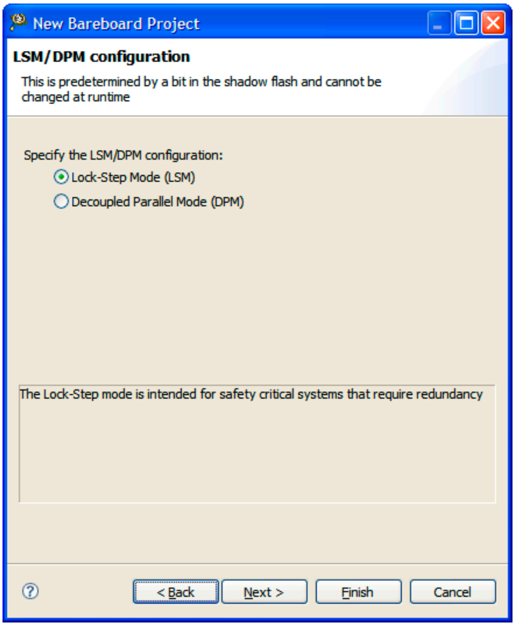
8. Click **Next**.

   The **Connections** page appears.

9. Check the appropriate connection. For example, check the **P&E USB MultiLink PPCNEXUS** checkbox.

   The **LSM/ DPM Configuration** page appears.

Figure 5.1 New Bareboard Project Wizard — LSM/ DPM Configuration Page



LSM/DPM configuration

This is predetermined by a bit in the shadow flash and cannot be changed at runtime

Specify the LSM/DPM configuration:

● Lock-Step Mode (LSM)
○ Decoupled Parallel Mode (DPM)

The Lock-Step mode is intended for safety critical systems that require redundancy

10. Select the **Lock-Step Mode (LSM)** option and click **Next**.

    The **Add Files** page appears.

11. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.

    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.
    b. Click **OK** to close the **Add File Path** dialog box.
    c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

   d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **Languages and Build Tools Options** page appears.

13. Select the appropriate options to enable programming language, build tools options, and floating point supports.

14. Click **Finish**.

    The wizard creates a project for the Kinetis architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Creating DPM Project

To create a DPM, perform these steps.

1. Select **Start > Programs > Freescale CodeWarrior > CW MCU V10.1 > CodeWarrior**.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

2. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

3. Click the **Go to Workbench** link.

   The **Workbench** window opens.

4. Select **File > New > Bareboard Project**, from the IDE menu bar.

   The **Create an MCU bareboard Project** page of the **New Bareboard Project** wizard appears.

5. Specify a name for the new project. For example, enter the project name as `Project_2.`

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

6. Click **Next**.

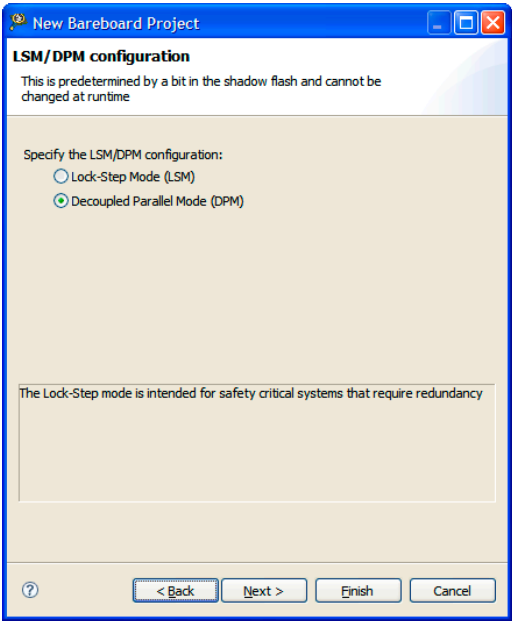
   The **Devices** page appears.

7. Expand the tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.
8. Click **Next**.

   The **Connections** page appears.
9. Check the appropriate connection. For example, check the **P&E USB MultiLink PPCNEXUS** checkbox.

   The **LSM/ DPM Configuration** page appears.

Figure 5.2 New Bareboard Project Wizard — LSM/ DPM Configuration Page



LSM/DPM configuration

This is predetermined by a bit in the shadow flash and cannot be changed at runtime

Specify the LSM/DPM configuration:
○ Lock-Step Mode (LSM)
● Decoupled Parallel Mode (DPM)

The Lock-Step mode is intended for safety critical systems that require redundancy

10. Select the **Decoupled Parallel Mode (DPM)** option and click **Next**.

    The **Add Files** page appears.
11. If you want to add a file to the project, click .

    The **Add File Path** dialog box appears.
    a. Specify the path of the file you want to add or browse the file by clicking the **File system** button.
    b. Click **OK** to close the **Add File Path** dialog box.

   c. Check the **Copy files into project** checkbox if you want to add the selected file in the project. If you clear the **Copy files into project** checkbox, the file is linked into the project and not copied.

   d. Clear the **Create main.c/main.asm file** checkbox if you do not want to create the main source file in the project.

12. Click **Next**.

    The **Languages and Build Tools Options** page appears.

13. Select the appropriate options to enable programming language, build tools options, and floating point supports.

14. Click **Finish**.

    The wizard creates a project for the Kinetis architecture. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

Debugging LSM/DPM Projects

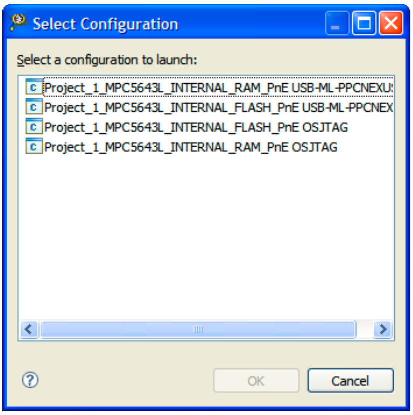
The settings in the launch configurations handles all core-specific initializations. To set up the launch configurations, perform these steps.

1. Ensure that you machine has USB NEXUS Multilink connected to a Leopard (MPC5643L) DPM board.

2. Click the **Debug** icon in the IDE's toolbar in the **Debu**g perspective.

   A menu of launch configurations appears.

Figure 5.3 Select Configurations Dialog Box



Select a configuration to launch:

Project_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXU
Project_1_MPC5643L_INTERNAL_FLASH_PnE USB-ML-PPCNEX
Project_1_MPC5643L_INTERNAL_FLASH_PnE OSJTAG
Project_1_MPC5643L_INTERNAL_RAM_PnE OSJTAG

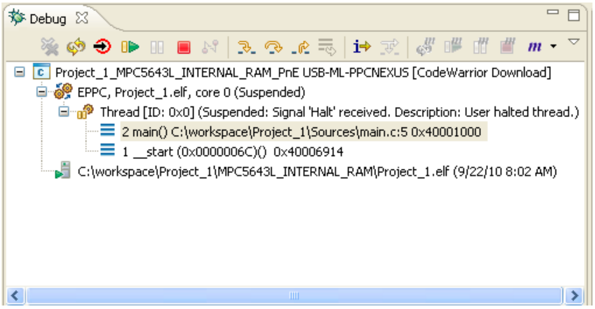
3. Select the desired launch configuration from this menu.

   The debugger downloads the selected core and switches to the **Debug** perspective.

4. Debugger halts execution at first statement of main().

   The Debug view displays all the threads associated with the core.

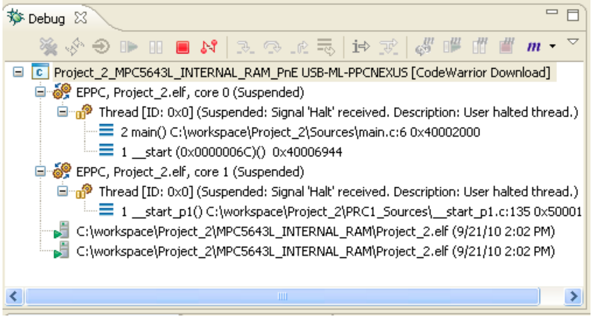
Figure 5.4 Multicore Debugging — Debug Core 0



```
Project_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS [CodeWarrior Download]
  EPPC, Project_1.elf, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() C:\workspace\Project_1\Sources\main.c:5 0x40001000
      1 __start (0x0000006C)()  0x40006914
  C:\workspace\Project_1\MPC5643L_INTERNAL_RAM\Project_1.elf (9/22/10 8:02 AM)
```

5. Repeat steps one through four to download all the other cores.

Figure 5.5 Multicore Debugging — Debug All Cores



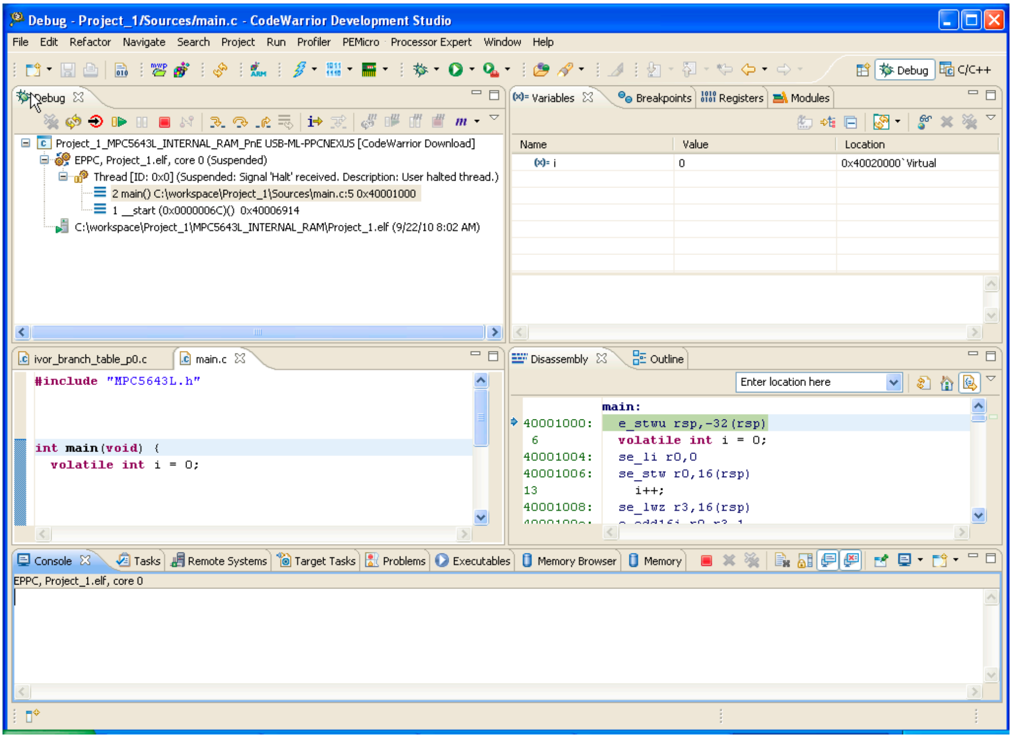
```
Project_2_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS [CodeWarrior Download]
  EPPC, Project_2.elf, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() C:\workspace\Project_2\Sources\main.c:6 0x40002000
      1 __start (0x0000006C)()  0x40006944
  EPPC, Project_2.elf, core 1 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      1 __start_p1() C:\workspace\Project_2\PRC1_Sources\__start_p1.c:135 0x50001
  C:\workspace\Project_2\MPC5643L_INTERNAL_RAM\Project_2.elf (9/21/10 2:02 PM)
  C:\workspace\Project_2\MPC5643L_INTERNAL_RAM\Project_2.elf (9/21/10 2:02 PM)
```

NOTE When you enter debug, the DPM project will show you two cores, first stopped at main() and the second stopped at __start_p1. For the LSM project you will see only one core stopped at main().

6. Select a thread from core 0 in the **Debug** view.

   All the views in the Debug perspective will be updated to display the debug session for the selected core.

Figure 5.6 Viewing Debug Information for Core 0



Debug - Project_1/Sources/main.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run Profiler PEMicro Processor Expert Window Help
Debug
C/C++
Debug
Variables
Breakpoints
Registers
Modules
ivor_branch_table_p0.c
main.c
Disassembly
Outline
Enter location here
Console
Tasks
Remote Systems
Target Tasks
Problems
Executables
Memory Browser
Memory

```
Project_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS [CodeWarrior Download]
  EPPC, Project_1.elf, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() C:\workspace\Project_1\Sources\main.c:5 0x40001000
      1 __start (0x0000006C)()  0x40006914
  C:\workspace\Project_1\MPC5643L_INTERNAL_RAM\Project_1.elf (9/22/10 8:02 AM)
```

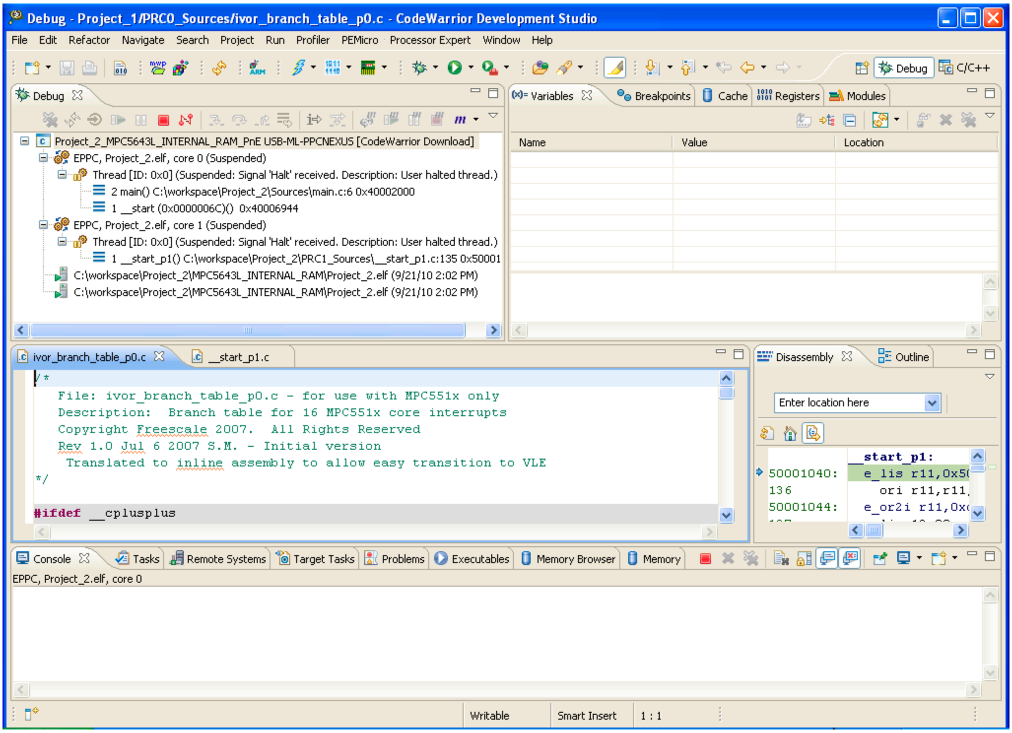
Name	Value	Location
i	0	0x40020000`Virtual

```
#include "MPC5643L.h"



int main(void) {
  volatile int i = 0;
```

```
          main:
40001000:   e_stwu rsp,-32(rsp)
 6          volatile int i = 0;
40001004:   se_li r0,0
40001006:   se_stw r0,16(rsp)
13            i++;
40001008:   se_lwz r3,16(rsp)
```

```
EPPC, Project_1.elf, core 0
```

Figure 5.7 Viewing Debug Information for Core 0


Debug - Project_1/PRC0_Sources/ivor_branch_table_p0.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run Profiler PEMicro Processor Expert Window Help
Debug
C/C++
Variables
Breakpoints
Cache
Registers
Modules
Name
Value
Location
ivor_branch_table_p0.c
__start_p1.c
Disassembly
Outline
Enter location here
Console
Tasks
Remote Systems
Target Tasks
Problems
Executables
Memory Browser
Memory
Writable
Smart Insert
1 : 1

```
Project_2_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS [CodeWarrior Download]
  EPPC, Project_2.elf, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() C:\workspace\Project_2\Sources\main.c:6 0x40002000
      1 __start (0x0000006C)() 0x40006944
  EPPC, Project_2.elf, core 1 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      1 __start_p1() C:\workspace\Project_2\PRC1_Sources\__start_p1.c:135 0x50001
  C:\workspace\Project_2\MPC5643L_INTERNAL_RAM\Project_2.elf (9/21/10 2:02 PM)
  C:\workspace\Project_2\MPC5643L_INTERNAL_RAM\Project_2.elf (9/21/10 2:02 PM)
```

```
/*
   File: ivor_branch_table_p0.c - for use with MPC551x only
   Description:  Branch table for 16 MPC551x core interrupts
   Copyright Freescale 2007.  All Rights Reserved
   Rev 1.0 Jul 6 2007 S.M. - Initial version
    Translated to inline assembly to allow easy transition to VLE
*/

#ifdef __cplusplus
```

```
            __start_p1:
50001040:    e_lis r11,0x5(
136            ori r11,r11,
50001044:    e_or2i r11,0x(
```

```
EPPC, Project_2.elf, core 0
```

Debugging Multicore Projects

The debugger in CodeWarrior Development Suite provides the facility to debug multiple processors using a single debug environment. The run control operations can be operated independently or synchronously. A common debug kernel facilitates multicore, run control debug operations for examining and debugging the interaction of the software running on the different cores on the system. This topic describes how the CodeWarrior debugger connects and downloads software to separate cores in a multicore processor and how you can choose to debug a program on a specific core.

NOTE This procedure assumes that you have already created a Multicore project named `Project_1`.

To debug multicore projects, perform these steps.

- Targeting Core
- Starting Debugging Session for Core
- Debugging Specific Core
- Multicore Operations

Targeting Core

The CodeWarrior debugger connects to specific processor core through information stored in a Download launch configuration. Specifically, the core index value on the **Main** tab of the **Debug Configurations** dialog box determines the core targeted for debug operations.

NOTE The core index value starts from zero. That is, the first processor core has an index value of zero, the second processor core has an index of 1, and so on.

You can set this core index value in two locations:

- On the **Launch Configurations** page when using the New Project wizard to build a project. You choose the core index value on this page, and when you finish with the wizard, the value is set in a Download launch configuration that the wizard generates for you.
- On the **Main** tab on the **Debug Configurations** dialog box, when you are modifying the settings of a Download launch configuration.

Starting Debugging Session for Core

To connect the debugger to a specific core and start a debugging session, start the Download launch configuration for that core. You start a Download launch configuration from either the **Run** menu or the toolbar's **Debug** icon.

- From Debug Configurations Dialog Box
- From Run Menu
- From Toolbar's Debug Icon

From Debug Configurations Dialog Box

1. Select **Run > Debug Configurations**.

   The **Debug Configurations** dialog box appears.

2. Pick the desired launch configuration from the left-hand side panel.

From Run Menu

1. Choose **Run > Debug History**.

   A menu of launch configurations appears.

2. Select the desired launch configuration from this menu.

From Toolbar's Debug Icon

1. Click the **Debug** icon in the IDE's toolbar in the **Debu**g perspective.

   A menu of launch configurations appears.

2. Select the desired launch configuration from this menu.

Debugging Specific Core

After you select the launch configuration, the debugger downloads the program to the specific core and the **Debug** perspective appears. Within the **Debug** view, the program's thread appears. The thread is identified by its launch configuration name and the index value of the core that it executes on. If you are source code debugging, the program's source appears in an editor view.

To load software into more processor cores, choose their launch configurations from the Run menu or Debug icon. Once the software downloads, the threads for these programs subsequently appear in the Debug view. The source to these programs also appear in the editor view. To switch between the various source files, click on the tab that lists the source file's name.

NOTE Ensure that you select the same remote system for all the cores you want to debug. Also specify the respective cores initialization file in the Initialization tab of the remote system. Please refer to Tutorial: Debugging a Multicore Processor for more details.

To debug a specific core's program, click on its thread in the **Debug** view. The **Debug** perspective automatically displays the source, registers, and variables for this core. If you click on another thread, the **Debug** perspective updates all of the views to display that core's context.

Multicore Operations

This topic explains the various features available to you when debugging a multicore processor.

- Multicore Commands in CodeWarrior IDE
- Multicore Commands in Debugger Shell

Multicore Commands in CodeWarrior IDE

When you start a multicore debug session, multicore commands are enabled on the CodeWarrior IDE **Run** menu. These commands, when issued, affect all cores simultaneously. Table 5.1 describes each menu choice. For detailed information on these commands, refer the Freescale Eclipse Extension Guide.

Table 5.1 Multicore Debugging Commands

Command	Icon	Description
Multicore Resume	▶	Starts all cores of a Multicore system running simultaneously.
Multicore Suspend	⏸	Stops execution of all cores of a Multicore system simultaneously.
Multicore Restart	↻	Restarts all the debug sessions for all cores of a Multicore system simultaneously.
Multicore Terminate	■	Kills all the debug sessions for all cores of a Multicore system simultaneously.

NOTE The **Resume** and **Step Into** buttons are enabled whenever one or more selected cores are in the stopped state. The **Suspend** button is enabled whenever one or more selected cores are in the running state. The **Restart** and **Terminate** buttons are enabled whenever one or more cores are being debugged.

To use the Multicore commands from the **Debug** perspective, perform these steps.

1. Start a debugging session by selecting the appropriately configured launch configuration.
2. Click the **Debug** tab of the **Debug Configurations** dialog box.
3. If necessary, expand the desired core's list of active threads by clicking on the tree control.
4. Click on the thread you want to use with multicore operations.

NOTE Selecting a thread uses the Multicore Group the core is part of in the multicore. For more information on the Multicore Groups feature, refer to the Freescale Eclipse Extensions Guide.

5. From the **Run** menu, specify the multicore operation to perform on the thread.

NOTE The keyboard shortcut for the **Multicore Resume** operation is Shift-F8.

Multicore Commands in Debugger Shell

In addition to the multicore-specific toolbar buttons and menu commands available in the **Debug** view, the **Debugger Shell** has Multicore specific commands that can control the operation of one or more processor cores at the same time. Like the menu commands, the Multicore debugger shell commands allow you to select, start, and stop a specific core. You can also restart or kill sessions executing on a particular core. Table 5.2 lists and defines the affect of each Multicore debugging command.

Table 5.2 Multicore Debugging Commands

Command	Shortcut	Description
mc::go	mc::g	Resume multiple cores **Syntax** go Examples mc::go Resumes the selected cores associated with the current thread context.
mc::kill	mc::kill	Terminates the debug session for selected cores associated with the current thread context. **Syntax** mc::kill Examples `mc::kill` Terminates multiple cores.
mc::restart	mc::restart	Restarts the debug session for selected cores associated with the current thread context. **Syntax** mc::restart Examples `mc::restart` Restarts multiple cores

Table 5.2 Multicore Debugging Commands (*continued*)

Command	Shortcut	Description
mc::stop	mc::stop	Stops the selected cores associated with the current thread context. **Syntax** mc::stop Examples `mc::stop` Suspends multiple cores

Table 5.2 Multicore Debugging Commands (*continued*)

<table>
<tr><th>Command</th><th>Shortcut</th><th>Description</th></tr>
<tr><td>mc::group</td><td>mc::gr</td><td>Display or edit multicore groups
Syntax
group
group new <type-name> [<name>]
group rename <name>|<group-index> <new-name>
group remove <name>|<group-index> ...
group removeall
group enable|disable <index> ...|all
Examples
<code>mc::group</code>
Shows the defined groups, including indices for use in the <code>mc::group rename|enable|remove</code> set of commands.
<code>mc::group new 8572</code>
Creates a new group for system type <code>8572</code>. The group name will be based on the system name and will be unique. The enablement of the group elements will be all non-cores enabled, all cores disabled.
<code>mc::group rename 0 "My Group Name"</code>
Renames the group at index 0 to "My Group Name".
<code>mc::group enable 0 0.0</code>
Enables the group at index 0 and the element at index 0.0 of the mc::group command.
<code>mc::group remove "My Group Name"</code>
Removes the group named "My Group Name".
<code>mc::group removeall</code>
Removes all groups.</td></tr>
</table>

Table 5.2 Multicore Debugging Commands (*continued*)

Command	Shortcut	Description

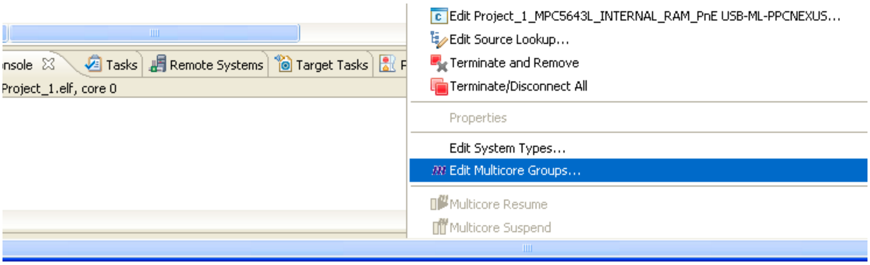
mc::type	mc::t	Shows the system types available for multicore debugging as well as type indices for use by the `mc::type remove` and `mc::group new` commands.
		Syntax
		type
		type import <filename>
		type remove <filename>|<type-index> ...
		type removeall
		Examples
		`mc::type`
		Display or edit system types
		`mc::type import 8572_jtag.txt`
		Creates a new type from the JTAG configuration file.
		`mc::type remove 8572_jtag.txt`
		Removes the type imported from the specified file.
		`mc::group removeall`
		Removes all imported types.

Editing Multicore Groups

To edit a multicore group, perform these steps.

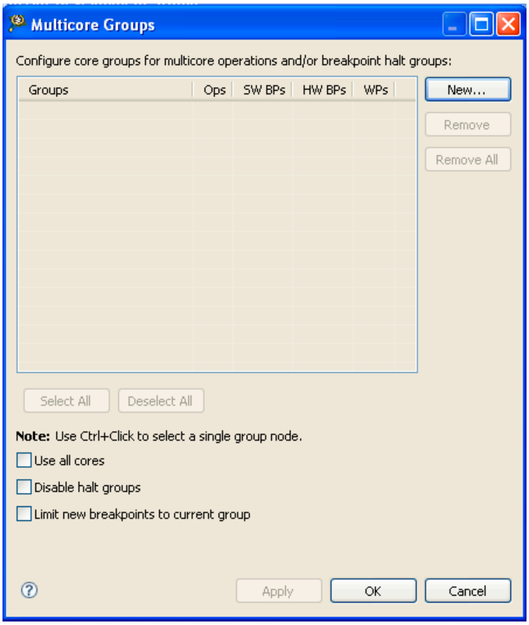
1. Right-click on the launch configuration in the **Debug** perspective and select **Edit Multicore Group** from the context menu, refer Figure 5.8.

Figure 5.8 Edit Multicore Group in Context Menu


nsole
Tasks
Remote Systems
Target Tasks
Project_1.elf, core 0
Edit Project_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS...
Edit Source Lookup...
Terminate and Remove
Terminate/Disconnect All
Properties
Edit System Types...
Edit Multicore Groups...
Multicore Resume
Multicore Suspend

The **Multicore Groups** dialog box appears, refer Figure 5.9.

Figure 5.9 Multicore Groups Dialog Box


Multicore Groups
Groups
Ops
SW BPs
HW BPs
WPs
New...
Remove
Remove All
Select All
Deselect All
Apply
OK
Cancel

Configure core groups for multicore operations and/or breakpoint halt groups:

Note: Use Ctrl+Click to select a single group node.

☐ Use all cores

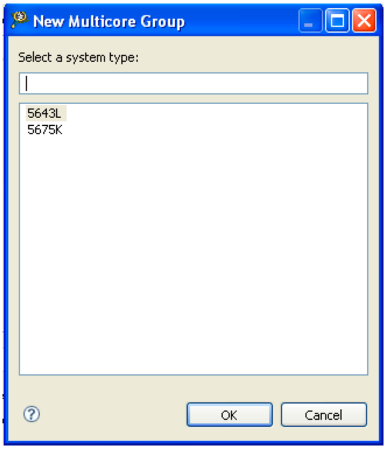
☐ Disable halt groups

☐ Limit new breakpoints to current group

2. Click **New**.

   The **New Multicore Group** dialog box appears, refer Figure 5.10. This is a simple selection dialog consisting of all types of multicore systems known to the product as well as any imported multiprocessor systems.

Figure 5.10 New Multicore Group Dialog Box

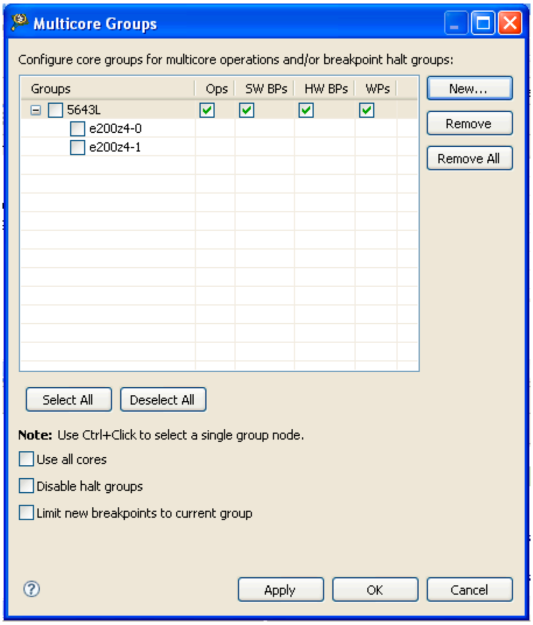


New Multicore Group
Select a system type:
5643L
5675K
OK
Cancel

3. Select a system type and click **OK**.

   The selected system type appears in the **Multicore Groups** dialog box, refer Figure 5.11.

Figure 5.11 Multicore Groups Dialog Box



Configure core groups for multicore operations and/or breakpoint halt groups:

Groups	Ops	SW BPs	HW BPs	WPs
⊟ ☐ 5643L	☑	☑	☑	☑
☐ e200z4-0				
☐ e200z4-1				

Note: Use Ctrl+Click to select a single group node.

☐ Use all cores

☐ Disable halt groups

☐ Limit new breakpoints to current group

The tree view in the dialog shows the list of defined groups. The groups are the top-level nodes in the tree, and each has an editable checkbox to specify whether the group is enabled or not. Each group contains one or more children, each representing a processor or a core. A processor will, in turn, have children representing the cores in the processor. Each processor/core node has an editable checkbox for choosing whether that element will be included in the group (root node).

The buttons are used as follows:

- **New** — Click to create a new group via the New Multicore Group dialog. The initial name of the group will be the system type name unless the name is already in use, in which case an index will be appended as is customary, e.g., '8572 (1)'. The initial enablement of the group and its descendants will be non-cores enabled, cores disabled. This guarantees an initial state with no error due to overlap.
- **Remove** — Click to remove the selected group. The button is disabled if no groups exist in the list or if no group is selected.
- **Remove All** — Click to remove all groups. The button is disabled if no groups exist in the list.
- **Select All** — Click to enable all groups, processors and cores. The button is disabled if no groups exist in the list.

- **Deselect All** — Click to disable all groups, processors and cores. The button is disabled if no groups exist in the list.

NOTE The buttons **Select All** and **Deselect All** are used as is customary in other dialog boxes to mean **Enable/Disable All Checkboxes**.

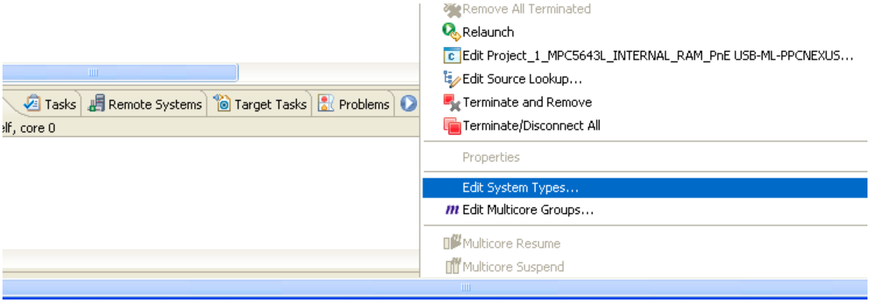
4. Click **Apply**.
5. Click **OK**.

Editing System Types

To edit a system type, perform these steps.

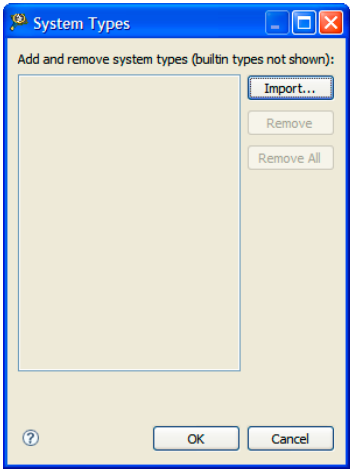
1. Right-click on the launch configuration in the **Debug** perspective and select **Edit System Types** from the context menu, refer Figure 5.12.

Figure 5.12 Edit System Types in Context Menu


Tasks
Remote Systems
Target Tasks
Problems
elf, core 0
Remove All Terminated
Relaunch
Edit Project_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS...
Edit Source Lookup...
Terminate and Remove
Terminate/Disconnect All
Properties
Edit System Types...
Edit Multicore Groups...
Multicore Resume
Multicore Suspend

The **System Types** dialog box appears, refer Figure 5.13.

Figure 5.13 System Types Dialog Box


System Types
Add and remove system types (builtin types not shown):
Import...
Remove
Remove All
OK
Cancel

2. Click **Import**.

   The **Import System Type** dialog box appears.

3. Select the desired file and click **Open**.

   The selected file appears in the **Import System Type** dialog box.

   The **Import System Type** dialog box lists the current imports, with buttons to remove existing entries and to import new entries. If you attempt to remove a type that has associated groups, you will be asked whether you want to continue, in which case all associated groups will be removed as well. Similarly, if you try to remove a type that has associated launch configurations, you will be asked whether you want to continue, in which case all associated launch configurations will be reset to the uniprocessor type.

6

Memory Management Unit Viewer

Eclipse CodeWarrior PowerPC e200 debugger provides access to the TLB (Translation Lookaside Buffer) array elements via a set of 128-bit register interface. Each 128-bit TLB register, corresponding to one TLB entry, is constructed from multiple MMU Assist registers.

You can inspect the MMU configuration status by reading the TLB register values from the register view or the register details view during the debug session. However, the MMU data in the TLB register view are displayed in a raw form and it is very time consuming and error-prone to manually decode the 128-bit TLB register value. The register detail view makes this decoding process easier by providing users with automatic bit field manipulations and register description text, but the MMU data from this view is still presented in a low level, requiring additional manual processing. Hence, there is a necessity of having a viewing tool that can display the complex TLB data in a more organized and structured format, i.e., the MMU viewer that processes the TLB register bits to a more easy-to-understand graphic display and permits on-the-fly view of the status of the current MMU configuration.

This chapter has following sections:

- MMU Viewer Pages
- MMU View

MMU Viewer Pages

This section explains each page of the **MMU Viewer**. You use these pages to configure MMU mapping. The tabbed interface of the **MMU Viewer** displays pages for configuration options and for generated code.

Table 6.1 lists the **MMU Viewer** pages.

Table 6.1 MMU Viewer Pages

Page	Description
TLB	This page displays the program memory-space mappings (virtual-to-physical address mappings) for the PowerPC e200.
TLB Table	This page provides a view of the entire configuration of all matrix entries in table form.

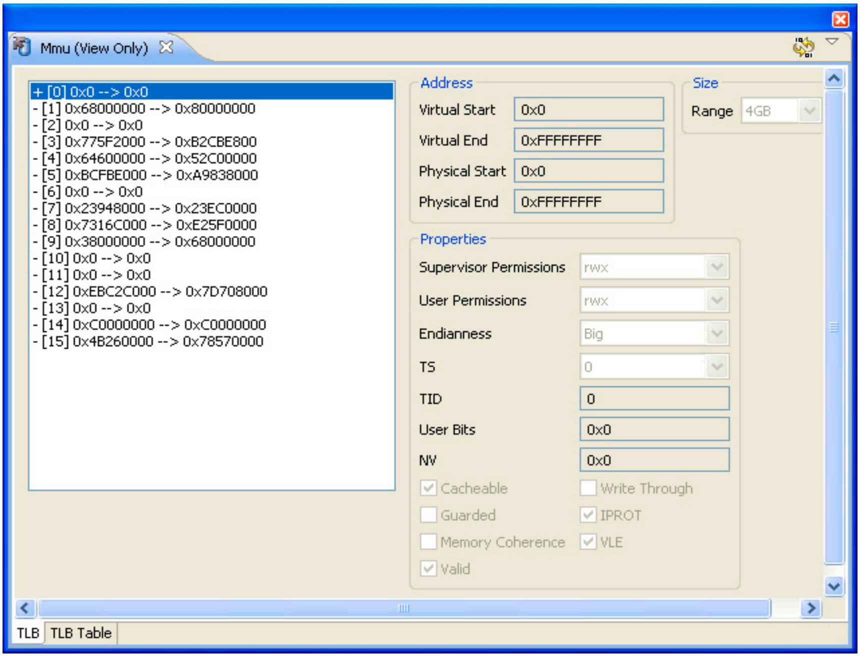
TLB

Use the **TLB** page to display program memory-space mappings (virtual-to-physical address mappings) for PowerPC e200. Each enabled and disabled memory-space mapping has a corresponding entry in the list on the left-hand side of the **TLB** page. The upper two groups, **Address** and **Size**, provide detailed memory-space mapping information for the selected TLB entry.

NOTE A plus sign to the left of an entry indicates an enabled mapping, and a minus sign indicates a disabled mapping.

Figure 6.1 shows the **TLB** page of the **MMU Viewer**.

Figure 6.1 MMU Viewer — TLB Page



```
+ [0] 0x0 --> 0x0
- [1] 0x68000000 --> 0x80000000
- [2] 0x0 --> 0x0
- [3] 0x775F2000 --> 0xB2CBE800
- [4] 0x64600000 --> 0x52C00000
- [5] 0xBCFBE000 --> 0xA9838000
- [6] 0x0 --> 0x0
- [7] 0x23948000 --> 0x23EC0000
- [8] 0x7316C000 --> 0xE25F0000
- [9] 0x38000000 --> 0x68000000
- [10] 0x0 --> 0x0
- [11] 0x0 --> 0x0
- [12] 0xEBC2C000 --> 0x7D708000
- [13] 0x0 --> 0x0
- [14] 0xC0000000 --> 0xC0000000
- [15] 0x4B260000 --> 0x78570000
```

Virtual Start 0x0
Virtual End 0xFFFFFFFF
Physical Start 0x0
Physical End 0xFFFFFFFF
Range 4GB
Supervisor Permissions rwx
User Permissions rwx
Endianness Big
TS 0
TID 0
User Bits 0x0
NV 0x0
☑ Cacheable ☐ Write Through
☐ Guarded ☑ IPROT
☐ Memory Coherence ☑ VLE
☑ Valid

Table 6.2 explains the options available on the **TLB** page of the **MMU Viewer**.

Table 6.2 MMU Viewer Settings

Option	Explanation
Virtual	The virtual base address of the data segment. This option corresponds to the Data Segment Virtual Base Address and Size (DSVBAS) bits of the Data Segment Descriptor Registers A (M_DSDAx) that configure the virtual base address.
Physical	The most-significant part of the physical address to use for translation. This option corresponds to the Data Segment Physical Base Address (DSPBA) bits of the Data Segment Descriptor Registers B (M_DSDBx).
Range	Entry’s page size. Supported page sizes are: 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB, 1GB, 4GB.

Table 6.2 MMU Viewer Settings

Option	Explanation
Supervisor Permissions	Select the access control that have to be provided to the supervisor. You can provide the controls to read (r), write (w), and execute (x) the instructions.
User Permissions	Select the access control that have to be provided to the user. You can provide the controls to read (r), write (w), and execute (x) the instructions.
Endianness	Select the option whether to access the page in big-endian byte order or in true little-endian byte order.
TS	Translation address space. This bit is compared with the IS or DS fields of the MSR (depending on the type of access) to determine if this TLB entry may be used for translation.
TID	Translation ID bits. This field is compared with the current process IDs of the effective address to be translated. A TID value of 0 defines an entry as global and matches with all process IDs.
User Bits	Select the user bits for use by the system software.
NV	Next replacement victim for TLB1 (software managed) Software updates this field; it is copied to the ESEL field on a TLB Error.
Cacheable	**Checked**—The page is considered cache-inhibited. **Cleared**—The page is considered cacheable.
Guarded	**Checked**—All loads and stores to this page are performed without speculation (that is, they are known to be required). **Cleared**—Access to this page are not guarded, and can be performed before it is known if they are required by the sequential execution model.
Memory Coherence	**Checked**— Memory coherence is required. **Cleared**—Memory coherence is not required.
Valid	**Checked**— The TLB entry is valid. **Cleared**—The TLB entry is invalid.
Write Through	**Checked**— All stores performed to this page are written through to main memory. **Cleared**—This page is considered write-back with respect to the caches in the system.

Table 6.2 MMU Viewer Settings

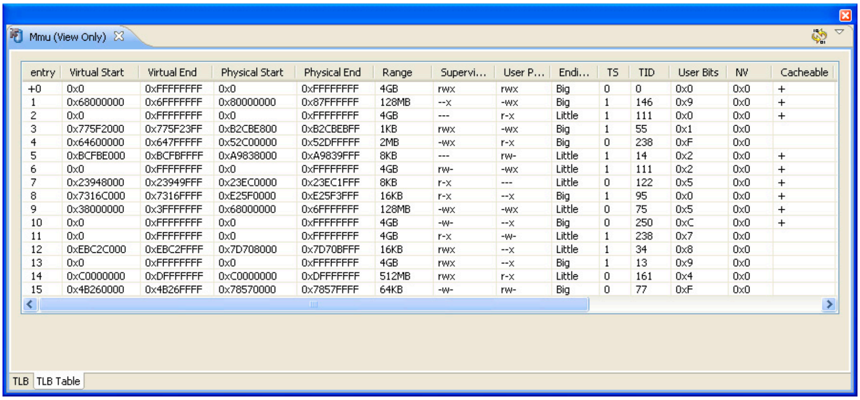
Option	Explanation
IPROT	**Checked**—The entry is protected from invalidation. **Cleared**—The entry is not protected from invalidation.
VLE	**Checked**— The page is a Power ISA VLE page. **Cleared**—The page is a standard Power VLE page.

TLB Table

The **TLB Table** page shows an alternate, tabular rendering of the settings that you specify on the **TLB** page. Use this page to view the configuration of all **TLB** mappings. MMU Viewer uses the TLB page settings to generate the column headers of this page. The table data shows the validated records for each **TLB** entry. You can resize the table columns to hide columns or view the larger data fields. A plus sign (**+**) in a table cell represents a checked checkbox in the associated **TLB** page.

Figure 6.2 shows the **TLB Table** page of the **MMU Viewer**.

Figure 6.2 MMU Viewer — TLB Table Page

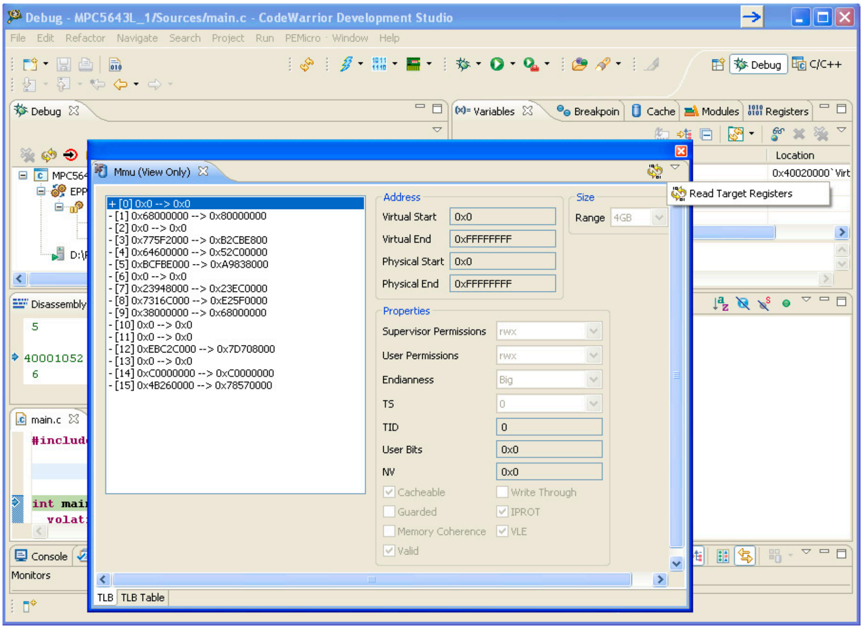


entry	Virtual Start	Virtual End	Physical Start	Physical End	Range	Supervi...	User P...	Endi...	TS	TID	User Bits	NV	Cacheable
+0	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	rwx	rwx	Big	0	0	0x0	0x0	+
1	0x68000000	0x6FFFFFFF	0x80000000	0x87FFFFFF	128MB	--x	-wx	Big	1	146	0x9	0x0	+
2	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	---	r-x	Little	1	111	0x0	0x0	+
3	0x775F2000	0x775F23FF	0xB2CBE800	0xB2CBEBFF	1KB	rwx	-wx	Big	1	55	0x1	0x0	
4	0x64600000	0x647FFFFF	0x52C00000	0x52DFFFFF	2MB	-wx	r-x	Big	0	238	0xF	0x0	
5	0xBCFBE000	0xBCFBFFFF	0xA9838000	0xA9839FFF	8KB	---	rw-	Little	1	14	0x2	0x0	+
6	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	rw-	-wx	Little	1	111	0x2	0x0	+
7	0x23948000	0x23949FFF	0x23EC0000	0x23EC1FFF	8KB	r-x	---	Little	0	122	0x5	0x0	+
8	0x7316C000	0x7316FFFF	0xE25F0000	0xE25F3FFF	16KB	r-x	--x	Big	1	95	0x0	0x0	+
9	0x38000000	0x3FFFFFFF	0x68000000	0x6FFFFFFF	128MB	-wx	-wx	Little	0	75	0x5	0x0	+
10	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	-w-	--x	Big	0	250	0xC	0x0	+
11	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	r-x	-w-	Little	1	238	0x7	0x0	
12	0xEBC2C000	0xEBC2FFFF	0x7D708000	0x7D70BFFF	16KB	rwx	--x	Little	1	34	0x8	0x0	
13	0x0	0xFFFFFFFF	0x0	0xFFFFFFFF	4GB	rwx	--x	Big	1	13	0x9	0x0	
14	0xC0000000	0xDFFFFFFF	0xC0000000	0xDFFFFFFF	512MB	rwx	r-x	Little	0	161	0x4	0x0	
15	0x4B260000	0x4B26FFFF	0x78570000	0x7857FFFF	64KB	-w-	rw-	Big	0	77	0xF	0x0	

MMU View

MMU view can be used to examine the current state of a thread's MMU configuration during the course of the debugging session. You can also detach the **MMU** view into its own floating window and reposition the window as a shell within the desktop. You can refresh the MMU data from the **Read Target Registers** available on the top right corner of the MMU view (Figure 6.3).

Figure 6.3 Read Target Registers — MMU Viewer


Debug - MPC5643L_1/Sources/main.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run PEMicro Window Help
Debug
C/C++
Variables
Breakpoin
Cache
Modules
Registers
Location
0x40020000` Virt
Disassembly
main.c
Console
Monitors
Mmu (View Only)
Read Target Registers
Address
Size
Properties
TLB
TLB Table

```
+ [0] 0x0 --> 0x0
- [1] 0x68000000 --> 0x80000000
- [2] 0x0 --> 0x0
- [3] 0x775F2000 --> 0xB2CBE800
- [4] 0x64600000 --> 0x52C00000
- [5] 0xBCFBE000 --> 0xA9838000
- [6] 0x0 --> 0x0
- [7] 0x23948000 --> 0x23EC0000
- [8] 0x7316C000 --> 0xE25F0000
- [9] 0x38000000 --> 0x68000000
- [10] 0x0 --> 0x0
- [11] 0x0 --> 0x0
- [12] 0xEBC2C000 --> 0x7D708000
- [13] 0x0 --> 0x0
- [14] 0xC0000000 --> 0xC0000000
- [15] 0x4B260000 --> 0x78570000
```

Virtual Start 0x0
Virtual End 0xFFFFFFFF
Physical Start 0x0
Physical End 0xFFFFFFFF
Range 4GB
Supervisor Permissions rwx
User Permissions rwx
Endianness Big
TS 0
TID 0
User Bits 0x0
NV 0x0
☑ Cacheable ☐ Write Through
☐ Guarded ☑ IPROT
☐ Memory Coherence ☑ VLE
☑ Valid

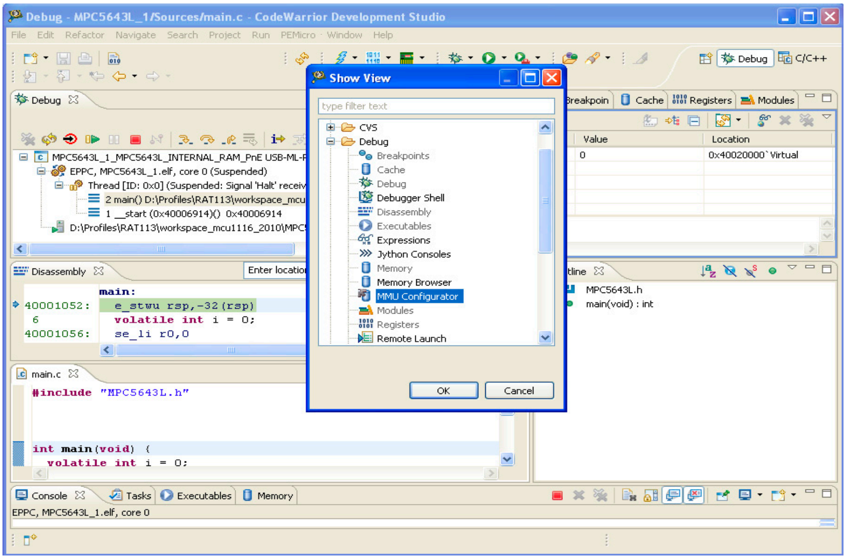
Opening MMU View

To open the **MMU** view, follow these steps:

1. Start a debugging session.
2. In the **Debug** view of the **Debug** perspective, select the process for which you want to work with MMU.
3. Select **Window > Show View > Other**.

   The **Show View** dialog box appears.
4. Expand the **Debug** tree control.
5. Select **MMU Configurator**(Figure 6.4).

Figure 6.4 Show View Dialog Box — MMU Configurator


Debug - MPC5643L_1/Sources/main.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run PEMicro Window Help
Debug
C/C++
Debug
Breakpoin
Cache
Registers
Modules
Value
Location
0
0x40020000` Virtual
Disassembly
Enter location
main.c
Outline
MPC5643L.h
main(void) : int
Show View
type filter text
CVS
Debug
Breakpoints
Cache
Debug
Debugger Shell
Disassembly
Executables
Expressions
Jython Consoles
Memory
Memory Browser
MMU Configurator
Modules
Registers
Remote Launch
OK
Cancel
Console
Tasks
Executables
Memory

MPC5643L_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-P
EPPC, MPC5643L_1.elf, core 0 (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt' receiv
2 main() D:\Profiles\RAT113\workspace_mcu
1 __start (0x40006914)() 0x40006914
D:\Profiles\RAT113\workspace_mcu1116_2010\MPC

```
         main:
40001052:  e_stwu rsp,-32(rsp)
 6         volatile int i = 0;
40001056:  se_li r0,0
```

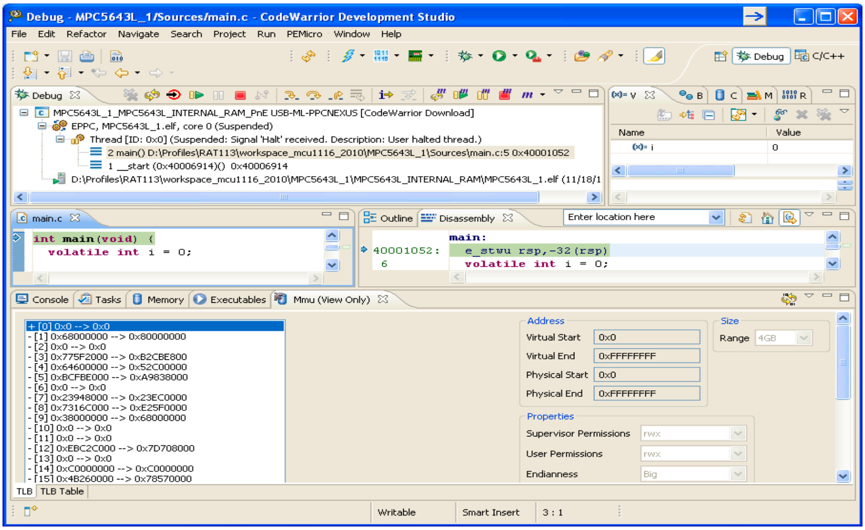
```
#include "MPC5643L.h"


int main(void) {
  volatile int i = 0;
```

EPPC, MPC5643L_1.elf, core 0

6. Click **OK**.

   The **Show View** dialog box closes. The **MMU** view appears, attached to an existing collection of views in the current perspective. Figure 6.5 shows the MMU view in Debug perspective.

Figure 6.5 MMU View


Debug - MPC5643L_1/Sources/main.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run PEMicro Window Help
Debug
C/C++
Debug
(x)= V
B
C
M
R
Name
Value
main.c
Outline
Disassembly
Enter location here
Console
Tasks
Memory
Executables
Mmu (View Only)
TLB
TLB Table
Writable
Smart Insert
3 : 1

```
MPC5643L_1_MPC5643L_INTERNAL_RAM_PnE USB-ML-PPCNEXUS [CodeWarrior Download]
  EPPC, MPC5643L_1.elf, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Profiles\RAT113\workspace_mcu1116_2010\MPC5643L_1\Sources\main.c:5 0x40001052
      1 __start (0x40006914)()  0x40006914
  D:\Profiles\RAT113\workspace_mcu1116_2010\MPC5643L_1\MPC5643L_INTERNAL_RAM\MPC5643L_1.elf (11/18/1
```

(x)= i 0

```
int main(void) {
  volatile int i = 0;
```

```
              main:
40001052:       e_stwu rsp,-32(rsp)
 6              volatile int i = 0;
```

```
+ [0] 0x0 --> 0x0
- [1] 0x68000000 --> 0x80000000
- [2] 0x0 --> 0x0
- [3] 0x775F2000 --> 0xB2CBE800
- [4] 0x64600000 --> 0x52C00000
- [5] 0xBCFBE000 --> 0xA9838000
- [6] 0x0 --> 0x0
- [7] 0x23948000 --> 0x23EC0000
- [8] 0x7316C000 --> 0xE25F0000
- [9] 0x38000000 --> 0x68000000
- [10] 0x0 --> 0x0
- [11] 0x0 --> 0x0
- [12] 0xEBC2C000 --> 0x7D708000
- [13] 0x0 --> 0x0
- [14] 0xC0000000 --> 0xC0000000
- [15] 0x4B260000 --> 0x78570000
```

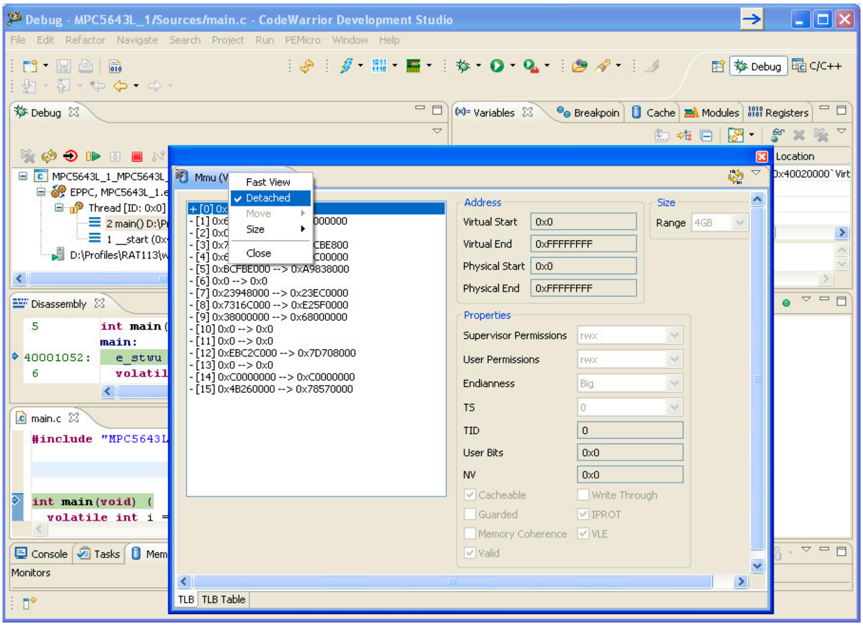
Address
Virtual Start 0x0
Virtual End 0xFFFFFFFF
Physical Start 0x0
Physical End 0xFFFFFFFF

Size
Range 4GB

Properties
Supervisor Permissions rwx
User Permissions rwx
Endianness Big

MMU View window can be undocked from any of the view docking areas by checking the **Detached** menu from the context menu of the tab. It helps you to view the MMU configuration and the thread window contents. Figure 6.6 shows the detached view of the MMU Viewer.

Figure 6.6 MMU Viewer Detached View


Debug - MPC5643L_1/Sources/main.c - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run PEMicro Window Help
Debug
C/C++
Variables
Breakpoin
Cache
Modules
Registers
Location
Mmu (V
Fast View
Detached
Move
Size
Close
Disassembly
main.c
Console
Tasks
Mem
Monitors
Address
Size
Properties
TLB
TLB Table

```
+ [0] 0x
- [1] 0x6          000000
- [2] 0x0
- [3] 0x7          CBE800
- [4] 0x6          C00000
- [5] 0xBCFBE000 --> 0xA9838000
- [6] 0x0 --> 0x0
- [7] 0x23948000 --> 0x23EC0000
- [8] 0x7316C000 --> 0xE25F0000
- [9] 0x38000000 --> 0x68000000
- [10] 0x0 --> 0x0
- [11] 0x0 --> 0x0
- [12] 0xEBC2C000 --> 0x7D708000
- [13] 0x0 --> 0x0
- [14] 0xC0000000 --> 0xC0000000
- [15] 0x4B260000 --> 0x78570000
```

Virtual Start 0x0
Virtual End 0xFFFFFFFF
Physical Start 0x0
Physical End 0xFFFFFFFF
Range 4GB
Supervisor Permissions rwx
User Permissions rwx
Endianness Big
TS 0
TID 0
User Bits 0x0
NV 0x0
☑ Cacheable ☐ Write Through
☐ Guarded ☑ IPROT
☐ Memory Coherence ☑ VLE
☑ Valid

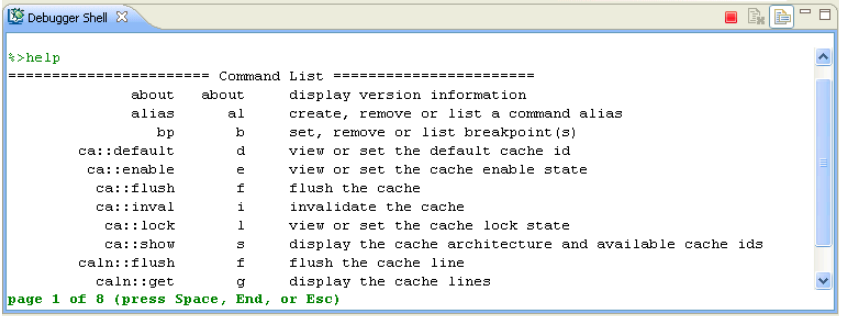
7

Scripting

CodeWarrior supports a command-line interface to some of its features including the debugger. You can use the command-line interface together with various scripting engines, such as the Microsoft® Visual Basic® script engine, the Java™ script engine, TCL, Python, and Perl. You can even issue a command that saves the command-line activity to a log file.

You can use the **Debugger Shell** view (Figure 7.1) to issue command lines to the IDE. For example, you can enter the command `debug` in this window to start a debugging session. The window lists the standard output and standard error streams of command-line activity.

Figure 7.1 Debugger Shell View



```
%>help
======================= Command List =======================
              about   about     display version information
              alias      al     create, remove or list a command alias
                 bp       b     set, remove or list breakpoint(s)
        ca::default       d     view or set the default cache id
         ca::enable       e     view or set the cache enable state
          ca::flush       f     flush the cache
          ca::inval       i     invalidate the cache
           ca::lock       l     view or set the cache lock state
           ca::show       s     display the cache architecture and available cache ids
        caln::flush       f     flush the cache line
          caln::get       g     display the cache lines
page 1 of 8 (press Space, End, or Esc)
```

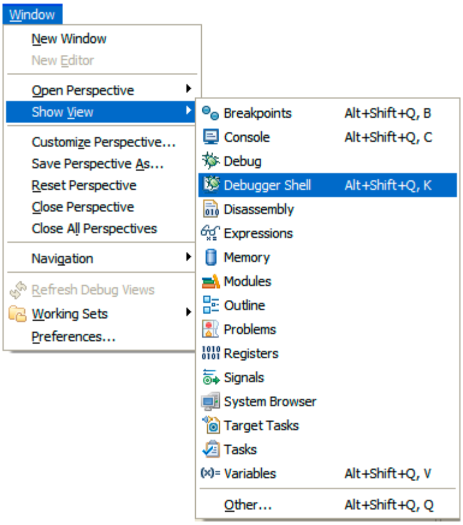
To open the **Debugger Shell** view, perform these steps.

1. Switch the IDE to the **Debug** perspective and start a debugging session.
2. Select **Window > Show View > Debugger Shell**.

   The **Debugger Shell** view appears (Figure 7.1).

NOTE Alternatively, select **Window > Show View > Other**. Expand the **Debug** tree control in the **Show View** dialog box, select **Debugger Shell**, and click **OK**.

Figure 7.2 Show View - Debugger Shell


Window
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective
Close Perspective
Close All Perspectives
Navigation
Refresh Debug Views
Working Sets
Preferences...
Breakpoints Alt+Shift+Q, B
Console Alt+Shift+Q, C
Debug
Debugger Shell Alt+Shift+Q, K
Disassembly
Expressions
Memory
Modules
Outline
Problems
Registers
Signals
System Browser
Target Tasks
Tasks
Variables Alt+Shift+Q, V
Other... Alt+Shift+Q, Q

To issue a command-line command, type the desired command at the command prompt (`%>`) in the **Debugger Shell** view, then press Enter or Return. The command-line debugger executes the specified command.

If you work with hardware as part of your project, you can use the command-line debugger to issue commands to the debugger while the hardware is running.

NOTE To list the commands the command-line debugger supports, type `help` at the command prompt and press Enter. The `help` command lists each supported command along with a brief description of each command.

TIP To view page-wise listing of the debugger shell commands, right-click in the **Debugger Shell** view and select **Paging** from the context menu. Alternatively, click the **Enable Paging** icon.

When you debug from the command line, you can use:

- Tcl Support
- Command-Line Debugging Tasks
- Debugger Shell Command List
- Microcontrollers-Specific HIWARE Commands

Tcl Support

This topic describes command-line debugger's Tcl support.

Resolution of Conflicting Command Names

The names of several command-line debugger commands conflict with the Tcl commands. Table 7.1 explains how the command-line debugger resolves such conflicts (if the mode is set to auto).

Table 7.1 Resolving Conflicting Commands

Command	Resolution
load	If you pass the command-line debugger a load command that includes a filename containing the suffix `.eld` or `.mcp`, the debugger loads the project. Otherwise, the debugger invokes the Tcl load command.
bp	If you pass the command-line debugger a `bp` command from within a script and the command has no arguments, the debugger invokes the `Tcl break` command. Otherwise, the debugger interprets a `break` command as a command to control breakpoints.
close	If you pass the command-line debugger a close command that has no arguments, the debugger terminates the debug session. Otherwise, the debugger invokes the Tcl close command.

Execution of Script Files

Tcl usually executes a script file as one large block, returning only after execution of the entire file. For the `run` command, however, the command-line debugger executes script files line-by-line. If a particular line is not a complete `Tcl` command, the debugger appends the next line. The debugger continues appending lines until it gets a complete Tcl script block.

Listing 7.1 lists code that includes a script. For the `Tcl source` command, the debugger executes this script as one block. But for the run debug command, the debugger executes this script as two blocks, the set statement and the while loop.

Listing 7.1 Example Tcl Script

```
set x 0;
while {$x < 5}
{

puts "x is $x";

set x [expr $x + 1]

}
```

NOTE The `run` debug command synchronizes debug events between blocks in a script file. For example, after a `go`, `next`, or `step` command, `run` polls the debug thread state and does not execute the next line or block until the debug thread terminates.

However, the Tcl `source` command does not consider the debug thread state. Consequently, use the `run` debug command to execute script files that contain these debug commands: `debug`, `go`, `next`, `stop`, and `kill`.

Tcl Startup Script

The command-line debugger can automatically run a Tcl script each time you open the command-line debugger window. This script is called a startup script.

You can use both Tcl and command-line debugger commands in the startup script. For example, you might include commands that set an alias or a define color configuration in a startup script.

To create a command-line debugger startup script, follow these steps.

1. Put the desired Tcl and command-line debugger commands in a text file.
2. Name this file `tcld.tcl`.
3. Place tcld.tcl in one of the directories listed below.
    - On a Windows® PC, put `tcld.tcl` in the system directory.

      For example, on Windows XP, put `tcld.tcl` in the WINDOWS directory.
    - On a Solaris Workstation, put `tcld.tcl` in your home directory.

NOTE There is no synchronization of debug events in the startup script. Consequently, put the `c` debug command to the startup script and place the debug commands `debug`, `go`, `stop`, `kill`, `next`, and `step` in another script so that they execute successfully.

Command-Line Debugging Tasks

Table 7.2 provides instructions for common command-line debugging tasks.

Table 7.2 Common Command-Line Debugging Tasks

Task	Instruction	Comments
Open the Debugger Shell	Select **Windows** > **Show View** > **Others** > **Debugger Shell**	The **Debugger Shell** view appears.
Use the help command	1. On the Debugger shell command prompt (**%>**), type `help`. 2. Press Enter.	The Command List for CodeWarrior is appears.
Enter a command	1. On the Debugger shell, type a command followed by a space. 2. Specify any valid command-line options, separating each with a space. 3. Press Enter.	You can use shortcuts instead of complete command names, such as k for kill.
View debug command hints	Specify alias followed by a space	The syntax for the rest of the command appears.
Review previous commands	Press Up Arrow and Down Arrow keys	
Clear command from the command line	Press the Esc key	
Stop an executing script	Press the Esc key	
Toggle between insert/overwrite mode	Press the Insert key	
Scroll up/ down a page	Press Page Up or Page Down key	

Table 7.2 Common Command-Line Debugging Tasks (*continued*)

Task	Instruction	Comments
Scroll left/right one column	Press Ctrl-Left Arrow or Ctrl-Right Arrow keys	
Scroll to beginning or end of buffer	Press Ctrl-Home or Ctrl-End keys	

Debugger Shell Command List

This topic lists and defines each command-line debugger command.

NOTE This chapter contains only the commands specific to the architectures supported in MCU product. The complete list of commands is available in *Freescale Eclipse Extensions Guide*.

cmdwin::eppc::getcoreid

Get the current core ID.

Syntax

```
cmdwin::eppc::getcoreid
```

Examples

```
cmdwin::eppc::getcoreid
```

Display the current core ID value.

cmdwin::eppc::setMMRBaseAddr

Set the MMR base address.

Syntax

```
cmdwin::eppc::SetMMRBaseAddr  <addr>
```

Examples

```
cmdwin::eppccmdwin::eppc::SetMMRBaseAddr 0x04700000
```

Tell the debugger to use 0x04700000 and the MMR base address.

All memory mapped register reads and writes will use this new base address.

cmdwin::eppc::setcoreid

Set the core ID.

Syntax

```
cmdwin::eppc::setcoreid  <coreID>
```

Examples

Table 7.3 lists and defines the examples of the `cmdwin::eppc::setcoreid` command.

Table 7.3 cmdwin::eppc::setcoreid Command-Line Debugger Command — Examples

Command	Description
`cmdwin::eppc::setcoreid 1`	Set the core ID value to 1.
`cmdwin::eppc::setcoreid default`	Set the core ID value to default. All commands will be executed on the new set core. To see the current core ID use `getcoreid` command.

gdi

Forwards third party custom commands.

Syntax

```
gdi<custom cmd> [<custom cmd arg list>]
```

Examples

```
gdi help
```

Forwards help command to currently selected GDI connection. The purpose of this command is to forward third party custom commands to their debug instrument, which is currently opened.

Microcontrollers-Specific HIWARE Commands

This topic lists and defines Microcontrollers-specific HIWARE commands.

Command List

Table 7.4 lists the supported HIWARE commands followed by:

- CodeWarrior debugger shell syntax,
- partially supported commands,
- commands that are not applicable in CodeWarrio,r
- commands supported in script files with TCL control flow statements, and
- unsupported commands.

The following columns represent the status in the CodeWarrior Eclipse IDE:

- *S-CW* — Command is supported followed by the CodeWarrior debugger shell syntax
- *P* — Command is partially supported, meaning some options/parameters are not supported
- *NA* — Command is not applicable in CodeWarrior
- *S-TCL* — Commands is supported in script files with TCL control flow statements
- *U* — Command is not supported

Table 7.4 Microcontrollers-Specific Debugger Command List

Command	Status	Description
HIWARE		
VER	*S-CW*	Lists the version of all loaded commands **Syntax** `about`
AUTOSIZE	*NA*	Selects window sizing mode
OPENIO	*NA*	Loads an IO simulation component

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
OPENPROJECT	*U*	Opens an existing project
OPEN	*NA*	Opens a component window
SET	*U*	Loads a target component
LOAD	*U*	Loads an application (Code & Symbols)
LOADCODE	*U*	Loads an application (Code only)
LOADSYMBOLS	*S-CW*	Loads an application (Symbols only) **Syntax** `loadsym <filename>`
FONT	*U*	Changes font in component windows
BCKCOLOR	*U*	Changes background color of component windows
SLAY	*NA*	Saves the layout and options of all components
ACTIVATE	*NA*	Activates a window component (in/out focus)
CLOSE	*NA*	Closes a component window
SYSTEM	*S-CW*	Executes an external application **Syntax** `system <command>`
EXIT	*S-CW*	Terminates this application **Syntax** `quitIDE`
RESET	*S-CW*	Resets the target MCU **Syntax** `Reset`

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description

HELP	*S-CW*	Lists available commands; to get help on a specific command, use the command followed by '?'
		Syntax
		`help`
		`help <command>`
		`<command> ?`

HIWARE Engine		

LF	*S-CW*	Opens a log file
		Syntax (for command)
		`log c <file>`
		Syntax (for session)
		`log s <file>`

NOLF	*S-CW*	Closes a log file
		Syntax (for command)
		`log off c`
		Syntax (for session)
		`log off s`

CR	*U*	Records all commands to a file
NOCR	*U*	Stops recording commands to a file

LOG	*S-CW*	Specifies items to be logged
		Syntax (for command)
		`log c <file>`
		Syntax (for session)
		`log s <file>`

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description

BS	*P*	Sets breakpoint
		Syntax
		`bp [-{hw|sw|auto}] {<func>|[<ms>:]<addr>|<file> <line> [<column>]}`
		`bp all|#<id>|<func>|<addr> enable|disable|{ignore <count>}`
		`bp #<id> cond <c-expr>`

SAVEBP	*U*	Saves breakpoints into a file
STEPINTO	*S-CW*	Step Into **Syntax** `step [asm|src] into`
STEPOUT	*S-CW*	Step out **Syntax** `step [asm|src] out`
STEPOVER	*S-CW*	Step over **Syntax** `step [asm|src] over`
RESTART	*S-CW*	Restart execution **Syntax** `restart`
DDEPROTOCOL	*U*	DDE Protocol options
DEFINEVALUEDLG	*U*	Opens a GUI to define a value for the symbol/ variable given as parameter

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
BC	*S-CW*	Clears breakpoint **Syntax** `bp all|#<id>|<func>|<addr> off`
BD	*S-CW*	Lists breakpoints **Syntax** `bp`
GO	*S-CW*	Starts execution (Go) **Syntax** `go`
STOP	*S-CW*	Stops execution (Halt) **Syntax** `stop`
P	*S-CW*	Executes an instruction (Flat step) **Syntax** `stepi`
T	*S-CW*	Executes CPU instructions **Syntax** `stepi`

Configuration Example

- `radix x`
- `config MemIdentifier 0`
- `config MemWidth 32`
- `config MemAccess 32`
- `config MemSwap off`

Note: These options apply only to the memory commands below.

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
WB	*S-CW*	Writes byte(s) into target memory **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value>`
WW	*S-CW*	Writes word(s) into target memory (2 bytes) **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value>`
WL	*S-CW*	Writes long(s) into target memory (4 bytes) **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value`
MS	*S-CW*	Writes byte(s) into target memory (same as WB) **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value>`
RD	*S-CW*	Lists registers **Syntax** `reg all`
RS	*S-CW*	Sets registers **Syntax** `reg <reg-spec>{..<reg>|#<n>} [-s|-ns] [%<conv>] =<value>`
MEM	*U*	Lists memory map
DASM	*S-CW*	Disassembles target memory **Syntax** `disassemble pc|[<ms>:]<addr> [<count>]`

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
DB	*S-CW*	Lists byte(s) from target memory **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] [-np]`
DW	*S-CW*	Lists words from target memory (2 bytes) **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] [-np]`
DL	*S-CW*	Lists long(s) from target memory (4 bytes) **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] [-np]`
CD	*S-CW*	Lists or changes directory **Syntax** `cd`
E	*S-CW*	Evaluates an expression and lists its result **Syntax** `evaluate [#<format>] [-l] [<var|expr>]`

A	*S-CW*	Evaluates an expression and assigns its result to an existing variable
		Syntax
		`var <var-spec> [-s|-ns] [%<conv>]=[evaluate [#<format>] [-l] [<var|expr>]]`
		Example
		`var myVar = [evaluate 1+1] - assigns value "2" to "myVar"`

PRINTF	*U*	Display a string on the window using printf like format
FPRINTF	*U*	Write a string to a file using fprintf like format

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
NB	*S-CW*	Changes or displays the default number base for the value of expressions **Syntax** `evaluate [#<format>] [-l] [<var|expr>]`
LS	*U*	Lists also global variables and procedures of the loaded application

SREC	*P*	Loads of Motorola S-records from a specified file
		Syntax
		`restore -h *.lod [[<ms>:]<addr>|+<offset>] [8bit|16bit|32bit|64bit]`
		`restore -b *.lod [<ms>:]<addr> [8bit|16bit|32bit|64bit]`

SAVE	*S-CW*	Saves a specified block of memory to a specified file in Motorola S-record format **Syntax** `save -h|-b [<ms>:]<addr>... <filename> [-a|-o] [8bit|16bit|32bit|64bit]`
PAUSETEXT	*NA*	Displays a modal message box for testing purpose
TESTBOX	*NA*	Displays a modal message box with a given string
REGFILE	*U*	Loads the I/O register descriptions from a 'register file'
REGBASE	*U*	Sets the base address of the on-chip I/O registers
ANDB	*U*	Bitwise-AND with target memory byte
ANDW	*U*	Bitwise-AND with target memory word (2 bytes)

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
ANDL	*U*	Bitwise-AND with target memory long (4 bytes)
NANDB	*U*	Bitwise-NAND with target memory byte
NANDW	*U*	Bitwise-NAND with target memory word (2 bytes)
NANDL	*U*	Bitwise-NAND with target memory long (4 bytes)
ORB	*U*	Bitwise-OR with target memory byte
ORW	*U*	Bitwise-OR with target memory word (2 bytes)
ORL	*U*	Bitwise-OR with target memory long (4 bytes)
NORB	*U*	Bitwise-NOR with target memory byte
NORW	*U*	Bitwise-NOR with target memory word (2 bytes)
NORL	*U*	Bitwise-NOR with target memory long (4 bytes)
EXORB	*U*	Bitwise-EXOR with target memory byte
EXORW	*U*	Bitwise-EXOR with target memory word (2 bytes)
EXORL	*U*	Bitwise-EXOR with target memory long (4 bytes)
MEMCOPY	*S-CW*	Copies the target memory
MEMBITCOPY	*S-CW*	Copies one bit from one memory address to another bit to another memory address **Syntax** `copy [<ms>:]<addr>[..<addr>|#<bytes>] [<ms>:]<addr>`
DEFINE	*S-CW*	Defines a symbol and associates a value **Syntax** `set varName ?value? <TCL command>`

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
UNDEF	*S-CW*	Removes a symbol definition **Syntax** `unset varName <TCL command>`
RETURN	*U*	Terminates the current command processing level
GOTO	*U*	Goes to the line following the label
GOTOIF	*U*	Goes to the line following the label if condition is TRUE
WHILE	*S-TCL*	Executes commands as long as the condition is true
FOR	*S-TCL*	Executes commands up to a predefined number of times
REPEAT	S-TCL	Executes commands until a certain condition is true
IF	S-TCL	Executes different command sections depending on the conditions
FOCUS	*NA*	Assigns a component as the destination for all subsequent commands
WAIT	*S-CW*	Waits by time tenths of a second **Syntax** `wait`
AT	*U*	Executes the next command at time (in ms)
CF	*S-CW*	Executes commands in the specified command file
CALL	*S-CW*	Executes commands in the specified command file **Syntax** `source`
Source		

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
SPC	*NA*	Highlights the statement corresponding to the code address
SMEM	*NA*	Highlights the statements corresponding to the code address range
SMOD	*NA*	Loads the corresponding module's source text
SPROC	*NA*	Highlights the statement of the procedure that is in the procedure chain
FOLD	*NA*	Hides source text for clearness at program block level
UNFOLD	*NA*	Exhibits the contents of folded source text blocks
SLINE	*NA*	Displays the line
FINDPROC	*NA*	Find the Procedure
FIND	*NA*	Searches an arbitrary pattern in the currently loaded source file
ATTRIBUTES	*NA*	Sets up the display
Assembly		
SPC	*NA*	Lists the specified address
SMEM	*NA*	Lists the specified address
ATTRIBUTES	*NA*	Sets up the display
Procedure		
ATTRIBUTES	*NA*	Sets up the display
Register		
ATTRIBUTES	*NA*	Sets up the display
Memory		
SPC	*NA*	Lists the address given as an argument
SMEM	*NA*	Lists the memory range given as an argument

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	Status	Description
SMOD	*NA*	Lists the first global variable of the module
FILL	*S-CW*	Fills a memory range with the given value **Syntax** `mem <addr-spec> [<range>] [-s|-ns] [%<conv>] =<value>`
UPDATERATE	*NA*	Sets the update rate
ATTRIBUTES	*NA*	Sets up the display
COPYMEM	*S-CW*	Copies a memory range to a specified location **Syntax** `copy [<ms>:]<addr>[..<addr>|#<bytes>] [<ms>:]<addr>`
SEARCHPATTERN	*NA*	Search a pattern in memory
REFRESHMEMORY	*S-CW*	After releasing caches, refreshes the memory **Syntax** `refresh`
Data		
SPROC	*NA*	Displays local or global variables of the procedure given as parameter
ADDXPR	*NA*	Adds a new expression in the data component
PTRARRAY	*NA*	Switches on or off the pointer as array displaying
SMOD	*NA*	Displays global variables of the module given as parameter
ZOOM	*NA*	Exhibits the member fields of structures by 'diving' into the structure
UPDATERATE	*NA*	Sets the update rate of the data component

Table 7.4 Microcontrollers-Specific Debugger Command List (*continued*)

Command	**Status**	**Description**
DUMP	*P*	Dumps the content of the data component to the command line **Syntax** `display`
ATTRIBUTES	*NA*	Sets up the display
REFRESHDATA	*S-CW*	After releasing caches, refreshes the display **Syntax** `refresh`
Command		
CLR	*S-CW*	Clears the Command window **Syntax** `cls`
ATTRIBUTES	*NA*	Sets up the display

8

Build Properties for Linux Project

This chapter explains build properties for Microcontrollers Linux projects. The Microcontrollers **New Linux/uClinux Application Project** wizard uses the information it gathers from you to set up the project's build and launch configurations.

A project's *build configuration* contains information on the tool settings used to make the program. For example, it describes the compiler and linker settings, and the files involved, such as source and libraries.

A project's *launch configuration* describes how the IDE starts the program, such as whether it executes by itself on a target, or under debugger control. Launch configurations also specify the core the program executes on (if the target processor has multiple cores).They also specify the connection interface and communications protocol that the debugger uses to control the environment that the program executes in.

NOTE The settings of the CodeWarrior IDE's build and launch configuration correspond to an object called a *target* made by the classic CodeWarrior IDE.

When the wizard completes its process, it generates launch configurations with names that follow the pattern ***projectname*** - ***configtype*** - ***targettype***, where:

- ***projectname*** represents the name of the project
- ***configtype*** represents the project's name, which usually describes the build configuration
- ***targettype*** represents the type of target software or hardware on which the launch configuration acts

For each launch configuration, you can specify build properties, such as:

- additional libraries to use for building code
- behavior of the compilers, linkers, assemblers, and other build-related tools
- specific build properties, such as the byte ordering of the generated code

The topics in this chapter are:

- Changing Build Properties
- Restoring Build Properties

- Build Properties for Linux/uClinux Project

Changing Build Properties

The Microcontrollers **New Linux/uClinux Application Project** wizard creates a set of build properties for the project. You can modify these build properties to better suit your needs.

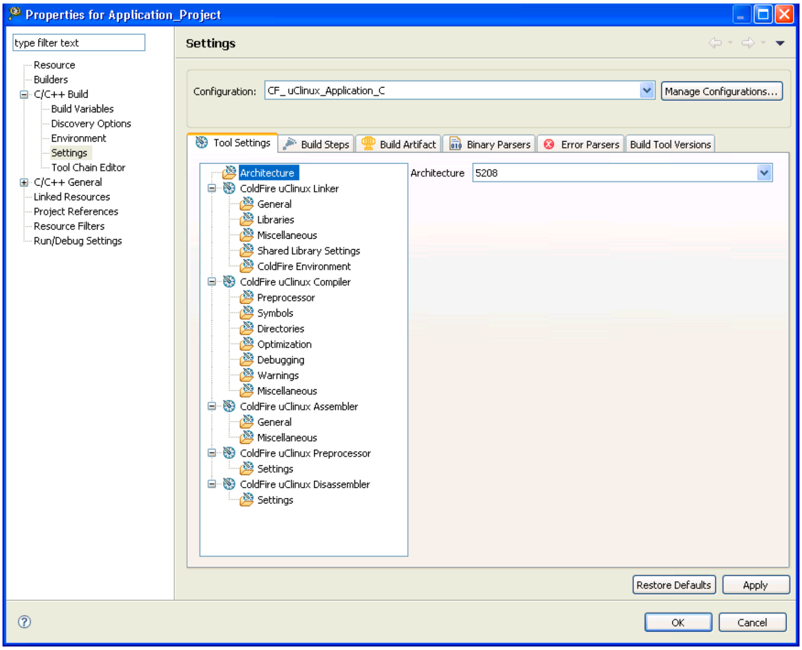
Perform these steps to change build properties:

1. Start the IDE.
2. In the **CodeWarrior Projects** view, select the project for which you want to modify the build properties.
3. Select **Project > Properties**.

   The **Properties** window appears. The left side of this window has a properties list. This list shows the build properties that apply to the current project.
4. Expand the **C/C++ Build** property.
5. Select **Settings**.

   The **Properties** window shows the corresponding build properties as in Figure 8.1.

Figure 8.1 Properties for <*Project*> Window


Properties for Application_Project
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Architecture 5208

6. Use the **Configuration** drop-down list to specify the launch configuration for which you want to modify the build properties.
7. Click the **Tool Settings** tab.

   The corresponding page appears.
8. From the list of tools on the **Tool Settings** page, select the tool for which you want to modify properties.
9. Change the settings that appear in the page.
10. Click **Apply**.

    The IDE saves your new settings.

You can select other tool pages and modify their settings. When you finish, click **OK** to save your changes and close the **Properties** window.

NOTE To build any Linux/uClinux application debug projects, ensure that the CodeSourcery GCCs are located on your machine. Usually they are located in the **Cross_Tools** folder in the CodeWarrior installation. Also, you must ensure that you have two environment variables defined for the two GCCs: Embedded Linux and uClinux. `CFGCCUCINSTALLDIR` referring to the `bin` folder of the `CodeSourcery uClinux GCC` install directory.

`CFGCCINSTALLDIR` referring to the `bin` folder of the `CodeSourcery GNU Linux GCC install` directory.

Restoring Build Properties

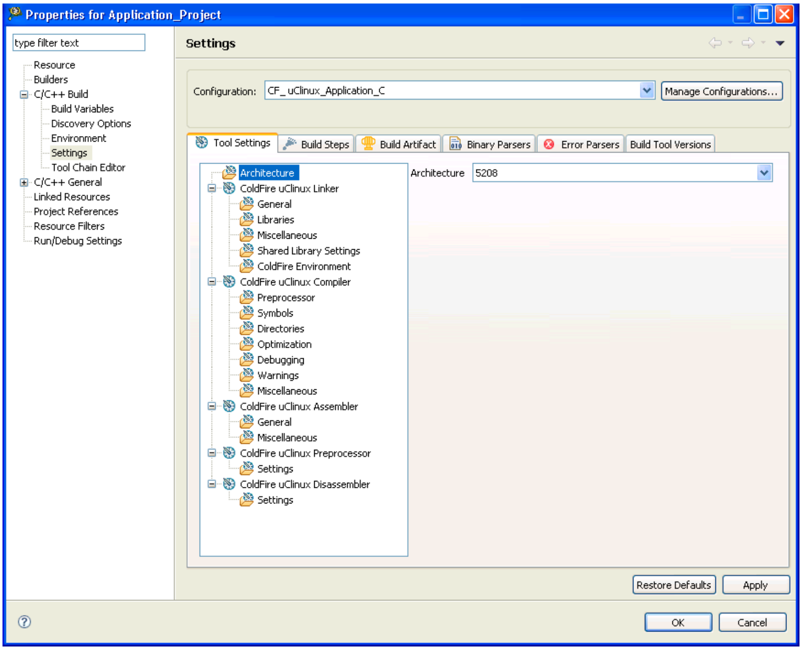
If you modify a build configuration that the CodeWarrior wizard generates, you can restore that configuration to its default state. You might want to restore the build properties in order to have a factory-default configuration, or to revert to a last-known working build configuration. To undo your modifications to build properties, click the **Restore Defaults** button at the bottom of the **Properties** window.

This changes the values of the options to the absolute default of the toolchain. By default, the toolchain options are blank.

Build Properties for Linux/uClinux Project

The **Properties for *<project>*** window shows the corresponding build properties for a uClinux project (Figure 8.2).

Figure 8.2 Build Properties — uClinux


Properties for Application_Project
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Architecture 5208

Table 8.1 lists the build properties specific to developing software for uClinux.

The properties that you specify in these panels apply to the selected build tool on the **Tool Settings** page of the **Properties for *<project>*** window.

Table 8.1 Build Properties for uClinux Project

Build Tool	Build Properties Panels
Architecture	Architecture
ColdFire uClinux Linker	ColdFire uClinux Linker > General
	ColdFire uClinux Linker > Libraries
	ColdFire uClinux Linker > Miscellaneous
	ColdFire uClinux Linker > Shared Library Settings
	ColdFire uClinux Linker > ColdFire Environment

Table 8.1 Build Properties for uClinux Project (*continued*)

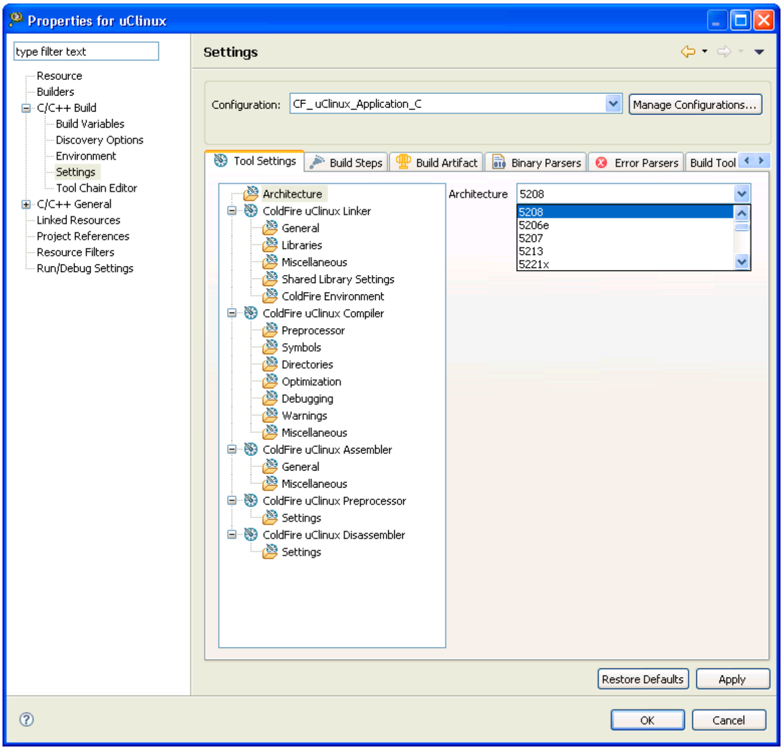
Build Tool	Build Properties Panels
ColdFire uClinux Compiler	ColdFire uClinux Compiler > Preprocessor
	ColdFire uClinux Compiler > Symbols
	ColdFire uClinux Compiler > Directories
	ColdFire uClinux Compiler > Optimization
	ColdFire uClinux Compiler > Debugging
	ColdFire uClinux Compiler > Warnings
	ColdFire uClinux Compiler > Miscellaneous
ColdFire uClinux Assembler	ColdFire uClinux Assembler > General
	ColdFire uClinux Assembler > Miscellaneous
ColdFire uClinux Preprocessor	ColdFire uClinux Preprocessor > Settings
ColdFire uClinux Disassembler	ColdFire uClinux Disassembler > Settings

Architecture

Use this panel to select the ColdFire uClinux architecture for the build.

Figure 8.3 shows the **Architecture** panel.

Figure 8.3 Tool Settings — Architecture



Configuration: CF_uClinux_Application_C

Architecture: 5208

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
5208
5206e
5207
5213
5221x
Restore Defaults
Apply
OK
Cancel

Table 8.2 lists and describes the options in the **Architecture** panel.

Table 8.2 Tool Settings — Architecture Options

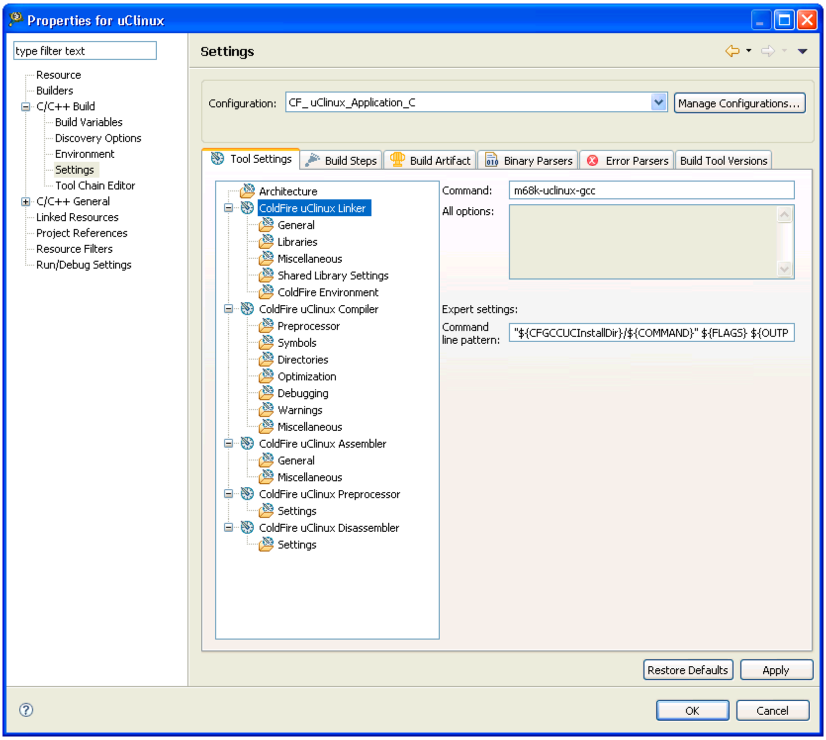
Option	Architecture
Architecture	Specify which architecture variant is used by the target.

ColdFire uClinux Linker

Use this panel to specify the command, options, and expert settings for the build tool linker. Additionally, the Linker tree control includes the general, libraries, and search path settings.

Figure 8.4 shows the **ColdFire uClinux Linker** panel.

Figure 8.4 Tool Settings — ColdFire uClinux Linker


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Command: m68k-uclinux-gcc

All options:

Expert settings:

Command line pattern: "${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTP

Table 8.3 lists and describes the options for the **ColdFire uClinux Linker** panel.

Table 8.3 tool Settings — ColdFire uClinux Linker Options

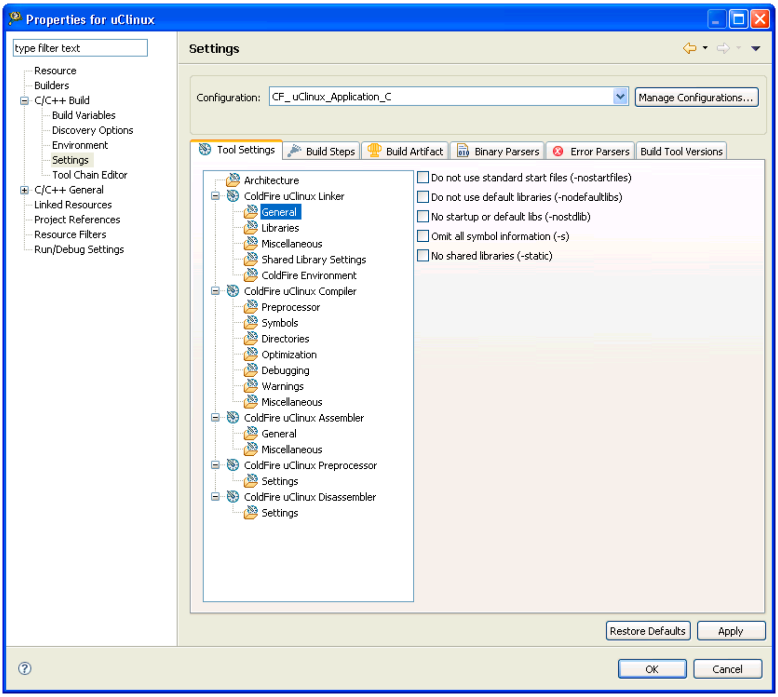
Option	Description
Command	Default: `m68k-uclinux-g++`
All options	Shows the actual command line the linker will be called with.
Expert settings	Default: "`${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`
Command line pattern	

ColdFire uClinux Linker > General

Use this panel to specify general settings for the ColdFire uClinux linker.

Figure 8.5 shows the **General** settings.

Figure 8.5 Tool Settings — ColdFire uClinux Linker > General



Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

☐ Do not use standard start files (-nostartfiles)
☐ Do not use default libraries (-nodefaultlibs)
☐ No startup or default libs (-nostdlib)
☐ Omit all symbol information (-s)
☐ No shared libraries (-static)

Table 8.4 lists and describes the general options for the **ColdFire uClinux Linker** panel.

Table 8.4 Tool Settings — ColdFire uClinux Linker > General Options

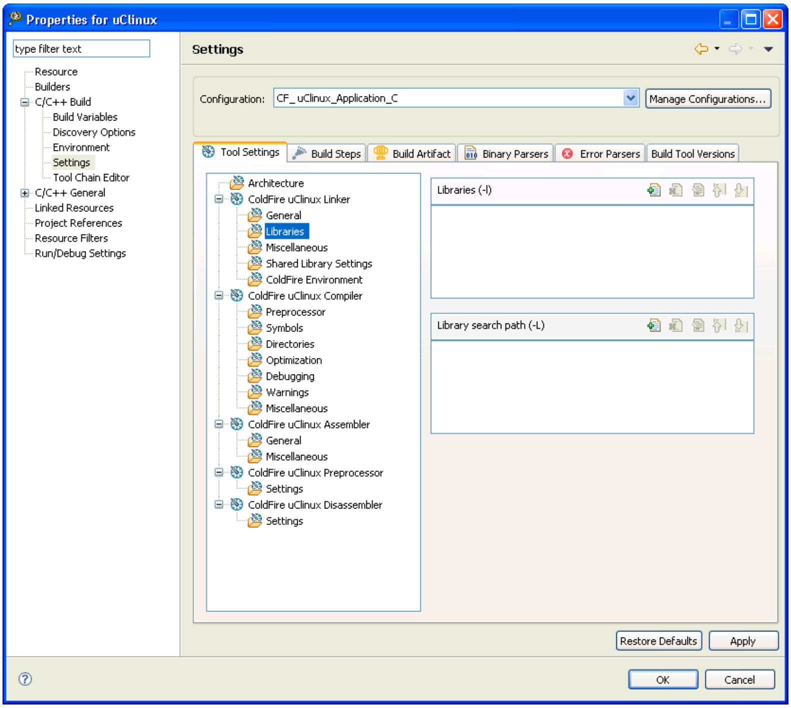
Option	Description
Do not use standard start files (-nostartfiles)	Check if you do not want to use the standard system startup files when linking. The standard system libraries are used by default, unless `-nostdlib` or `-nodefaultlibs` is used.
Do not use default libraries (-nodefaulltlibs)	Check if you do not want to use the standard system libraries when linking. Only the libraries you specify will be passed to the linker. The standard startup files are used normally, unless `-nostartfiles` is used. The compiler may generate calls to memcmp, memset, memcpy and memmove. These entries are usually resolved by entries in `libc`. These entry points should be supplied through some other mechanism when this option is specified.
No startup or default libs (-nostdlib)	Check if you do not want to use the standard system startup files or libraries when linking. No startup files and only the libraries you specify will be passed to the linker. The compiler may generate calls to `memcmp`, `memset`, `memcpy`, and `memmove`. These entries are usually resolved by entries in `libc`. These entry points should be supplied through some other mechanism when this option is specified. One of the standard libraries bypassed by `-nostdlib` and `-nodefaultlibs` is `libgcc.a`, a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages.
Omit all symbol information (-s)	Check if you do not want to remove all symbol table and relocation information from the executable.
No shared libraries (-static)	Check to prevent linking against or with the shared libraries for systems that support dynamic linking. On other systems, this option has no effect.

ColdFire uClinux Linker > Libraries

Use this panel to specify library settings for the **ColdFire uClinux Linker**. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 8.6 shows the **Libraries** settings.

Figure 8.6 Tool Settings — ColdFire uClinux Linker > Libraries



Configuration: CF_uClinux_Application_C

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Libraries (-l)
Library search path (-L)
Restore Defaults
Apply
OK
Cancel

Table 8.5 lists and describes the libraries options for the ColdFire uClinux linker.

Table 8.5 ColdFire uClinux Linker > Libraries

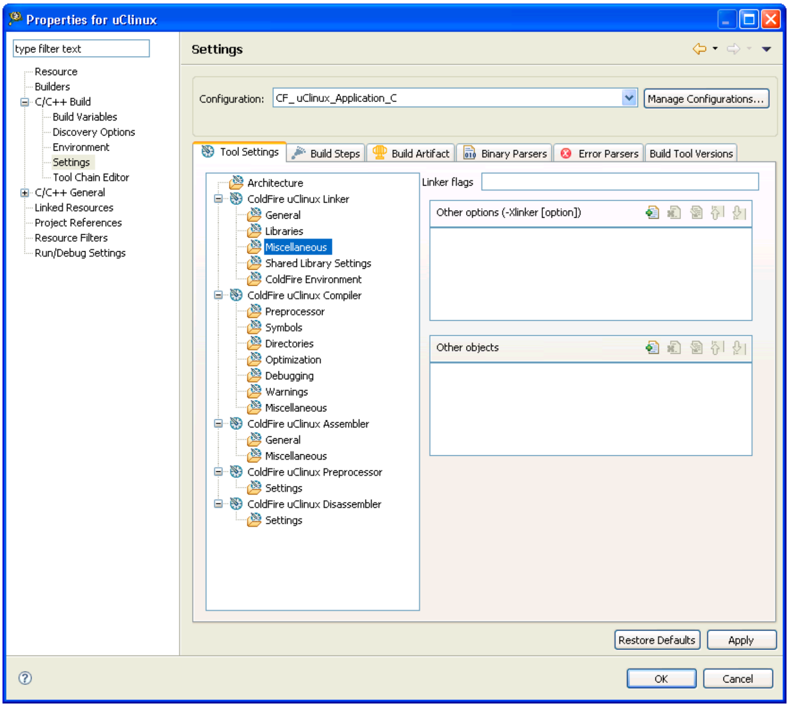
Option	Description
Libraries (-l)	Specifies the linker command-line switches for any libraries that you want the IDE to include in the GCC command line for each file in the project.
Library search path (-L)	Specifies the search pathname of libraries or other resources related to the project.

ColdFire uClinux Linker > Miscellaneous

Use this panel to specify miscellaneous settings for the **ColdFire uClinux Linker**.

Figure 8.7 shows the **Miscellaneous** settings.

Figure 8.7 Tool Settings — ColdFire uClinux Linker > Miscellaneous


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Linker flags
Other options (-Xlinker [option])
Other objects
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Table 8.6 lists and describes the miscellaneous options for **ColdFire uClinux Linker**.

Table 8.6 ColdFire uClinux Linker > Miscellaneous Options

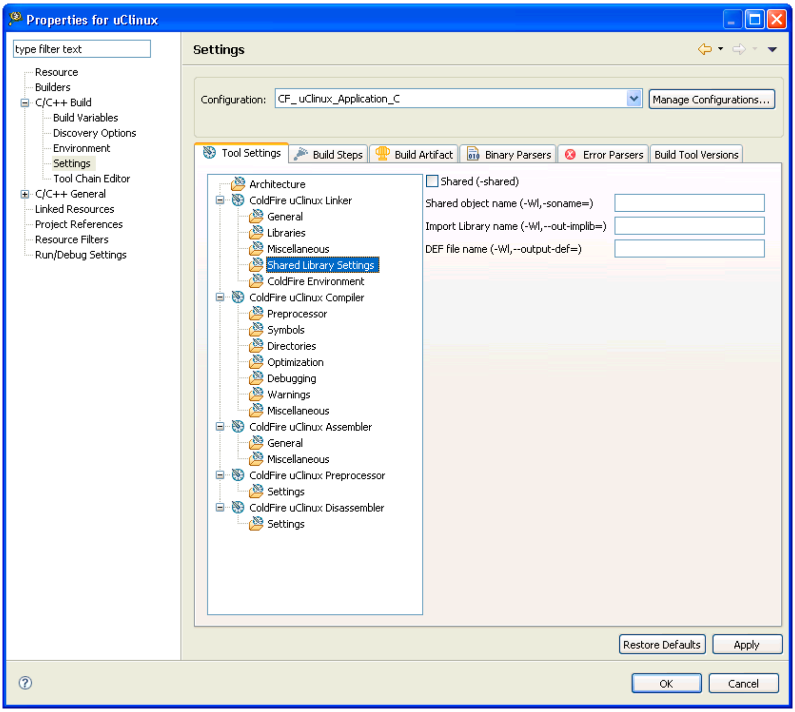
Option	Description
Linker flags	Specify the other required linker flag options.
Other options (-Xlinker [option])	Pass an option to the linker. You can use this option to supply system-specific linker options which GCC does not know how to recognize. If you want to pass an option that takes a separate argument, you must use `-Xlinker` twice, once for the option and once for the argument. For example, to pass `-assert definitions`, you must write `-Xlinker -assert -Xlinker definitions`. It does not work if you write `-Xlinker "-assert definitions` because this passes the entire string as a single argument, which is not what the linker expects. When using the GNU linker, it is usually more convenient to pass arguments to linker options using the `option=value` syntax than as separate arguments. For example, you can specify `-Xlinker -Map=output.map` rather than `-Xlinker -Map -Xlinker output.map`. Other linkers may not support this syntax for command-line options.
Other objects	Specify the other required object options.

ColdFire uClinux Linker > Shared Library Settings

Use this panel to specify shared library settings for the **ColdFire uClinux Linker**.

Figure 8.8 shows the **Shared Library Settings** panel.

Figure 8.8 Tool Settings — ColdFire uClinux Linker > Shared Library Settings


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

☐ Shared (-shared)

Shared object name (-Wl,-soname=)

Import Library name (-Wl,--out-implib=)

DEF file name (-Wl,--output-def=)

[Table 8.7](#) lists and describes the shared library settings for ColdFire uClinux linker.

Table 8.7 ColdFire uClinux Linker > Shared Library Settings Options

Option	Description
Shared (-shared)	Check to build shared versions of libraries, if shared libraries are supported on the target platform.
Shared object name (-Wl, -soname=)	Specifies the shared object name for the shared library.

Table 8.7 ColdFire uClinux Linker > Shared Library Settings Options (*continued*)

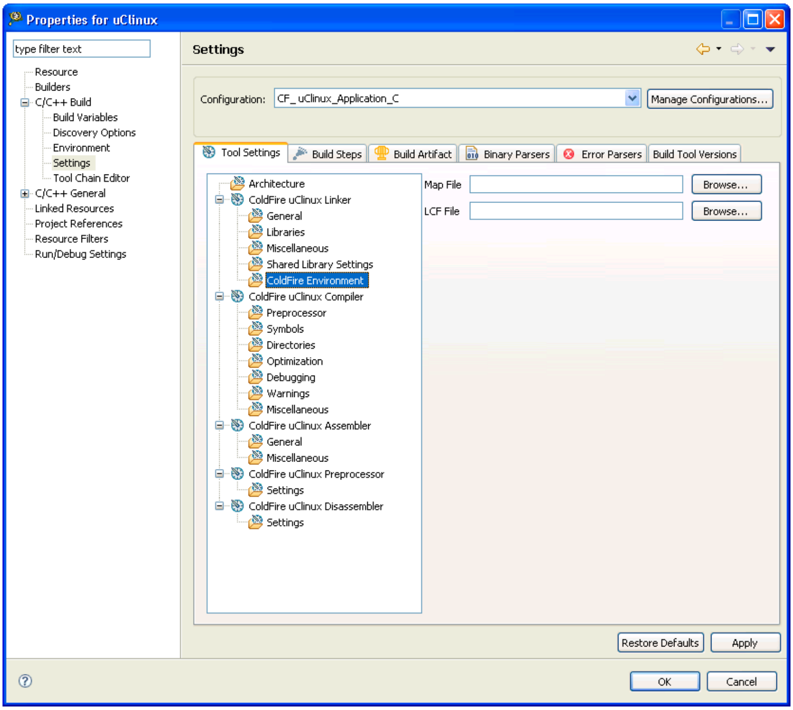
Option	Description
Import Library name (-Wl, --out-implib=)	Specifies the import library name. The linker will create the file which will contain an import lib corresponding to the DLL the linker is generating. This import lib (which should be called *.dll.a or *.a may be used to link clients against the generated DLL; this behaviour makes it possible to skip a separate dlltool import library creation step. This option is specific to the i386 PE targeted port of the linker.
DEF file name (_Wl, --output-def=)	Species the name of the `.def` file to be created by dlltool.

ColdFire uClinux Linker > ColdFire Environment

Use this panel to specify environment settings for the **ColdFire uClinux Linker**.

Figure 8.9 shows the **ColdFire Environment** settings.

Figure 8.9 Tool Settings — ColdFire uClinux Linker > ColdFire Environment Settings



Configuration: CF_uClinux_Application_C

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Map File
Browse...
LCF File
Browse...
Restore Defaults
Apply
OK
Cancel

Table 8.8 lists and describes the shared ColdFire environment settings for ColdFire uClinux linker.

Table 8.8 ColdFire uClinux Linker > ColdFire Environment Options

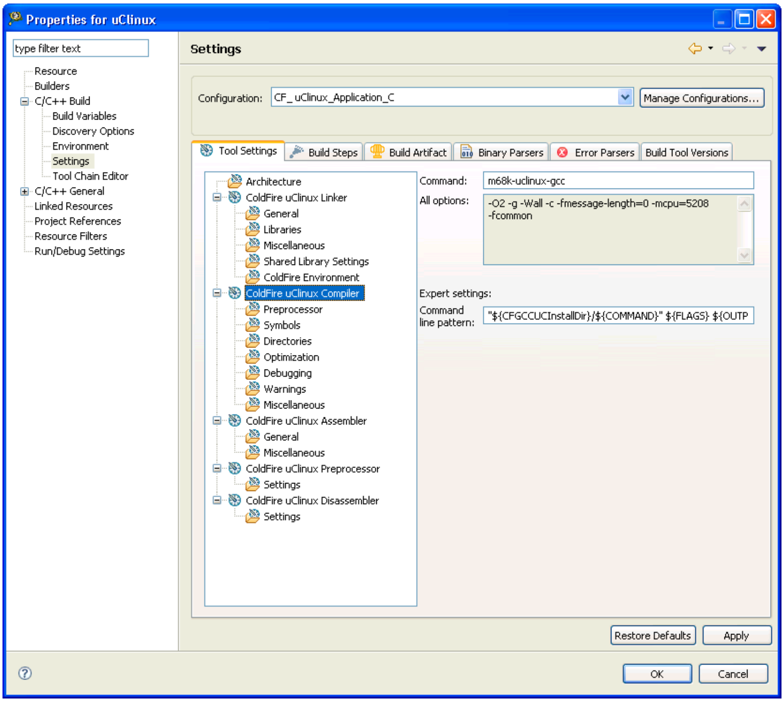
Option	Description
Map File	Specify the file that will be written as a linker memory map file.
LCF File	Read the specified linker script file.

ColdFire uClinux Compiler

Use this panel to specify the command, options, and expert settings for the build tool compiler. Additionally, the Compiler tree control includes the general, libraries, and search path settings.

Figure 8.10 shows the **ColdFire uClinux Compiler** panel.

Figure 8.10 Tool Settings — ColdFire uClinux Compiler


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Command: m68k-uclinux-gcc

All options: -O2 -g -Wall -c -fmessage-length=0 -mcpu=5208 -fcommon

Expert settings:
Command line pattern: "${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTP

Table 8.9 lists and describes the options for ColdFire uClinux compiler.

Table 8.9 Tool Settings — ColdFire uClinux Compiler Options

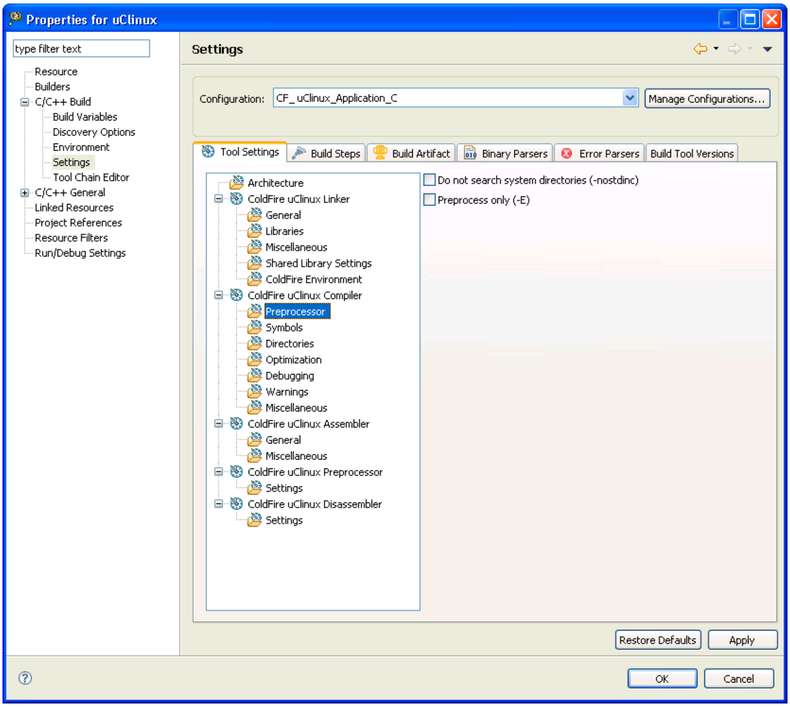
Option	Description
Command	Default: `m68k-uclinux-gcc`
All options	Shows the actual command line the compiler will be called with.
Expert settings	Default: `"${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`
Command line pattern	

ColdFire uClinux Compiler > Preprocessor

Use this panel to specify the preprocessor behavior. You can specify the file paths and define macros for the preprocessor.

Figure 8.11 shows the **ColdFire uClinux Compiler** panel.

Figure 8.11 Tool Settings — ColdFire uClinux Compiler > Preprocessor


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

☐ Do not search system directories (-nostdinc)

☐ Preprocess only (-E)

Table 8.10 lists and describes the options for the **ColdFire uClinux Compiler** panel.

Table 8.10 ColdFire uClinux Compiler > Preprocessor Options

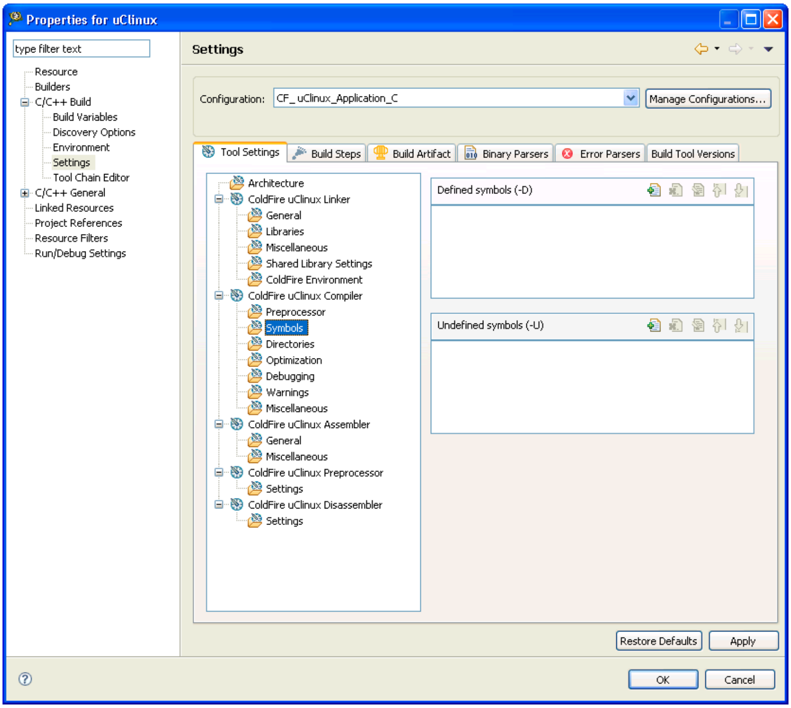
Option	Description
Do not search system directories (-nostdinc)	Select if you do not want to search the standard system directories for header files. Only the directories you have specified with '-I' options and the directory of the current file, if appropriate are searched.
Preprocess only (-E)	Select if you do not want to compile, assemble, or link.

ColdFire uClinux Compiler > Symbols

Use this panel to specify code- and symbol-generation options for the **ColdFire uClinux Compiler**.

Figure 8.12 shows the **ColdFire uClinux Compiler** panel.

Figure 8.12 Tool Settings — ColdFire uClinux Compiler > Symbols


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Defined symbols (-D)
Undefined symbols (-U)
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Table 8.11 lists and describes the options for the **ColdFire uClinux Compiler** panel.

Table 8.11 ColdFire uClinux Compiler > Symbols Options

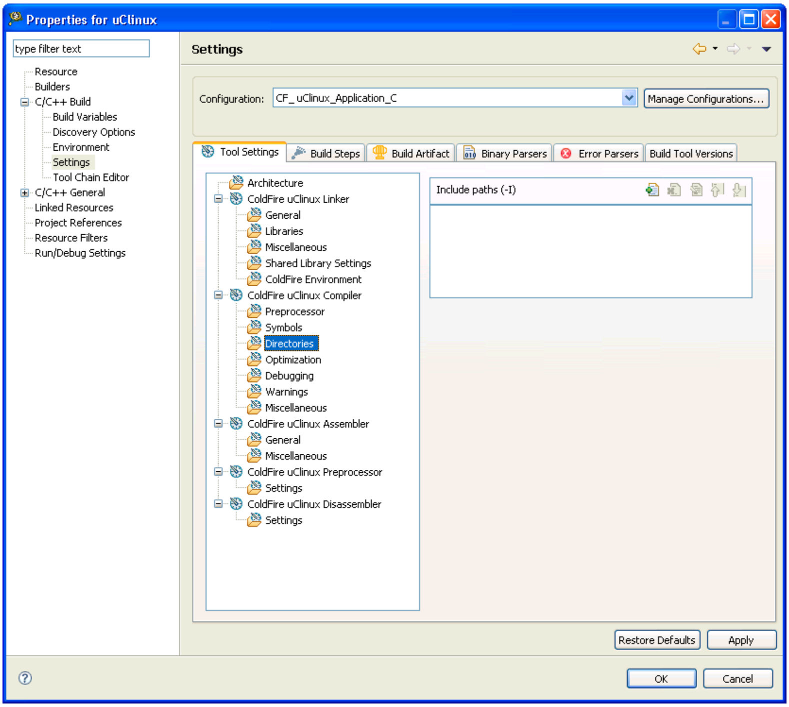
Option	Description
Defined symbols (-D)	Display only defined symbols for each object file.
Undefined symbols (-U)	Display only undefined symbols; those external to each object file.

ColdFire uClinux Compiler > Directories

Use this panel to specify include path directories for the **ColdFire uClinux Compiler**.

Figure 8.13 shows the **Directories** panel.

Figure 8.13 Tool Settings — ColdFire uClinux Compiler > Directories



Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Include paths (-I)
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Table 8.12 lists and describes the options for the **Directories** panel.

Table 8.12 ColdFire uClinux Compiler > Directories Options

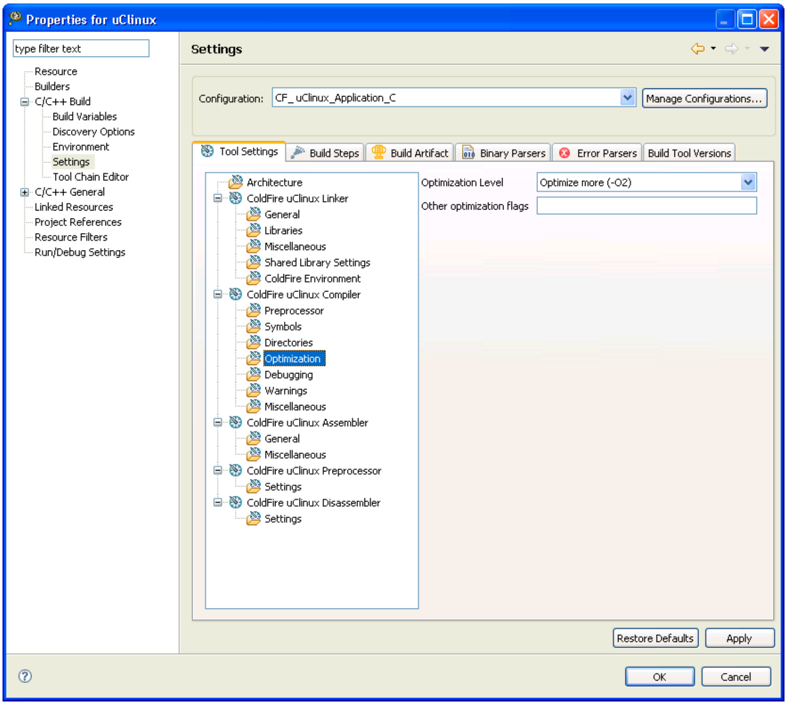
Option	Description
Include paths	Specify locations to header files. Append a list of directories to the standard directory list.

ColdFire uClinux Compiler > Optimization

Use this panel to control compiler optimizations. The compiler's optimizer can apply any of its optimizations in either global or non-global optimization mode. You can apply global optimization at the end of the development cycle, after compiling and optimizing all source files individually or in groups.

Figure 8.14 shows the **Directories** panel.

Figure 8.14 Tool Settings — ColdFire uClinux Compiler > Optimization


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Optimization Level: Optimize more (-O2)

Other optimization flags:

Table 8.13 lists and describes the options for the **Optimization** panel.

Table 8.13 ColdFire uClinux Compiler > Optimization Options

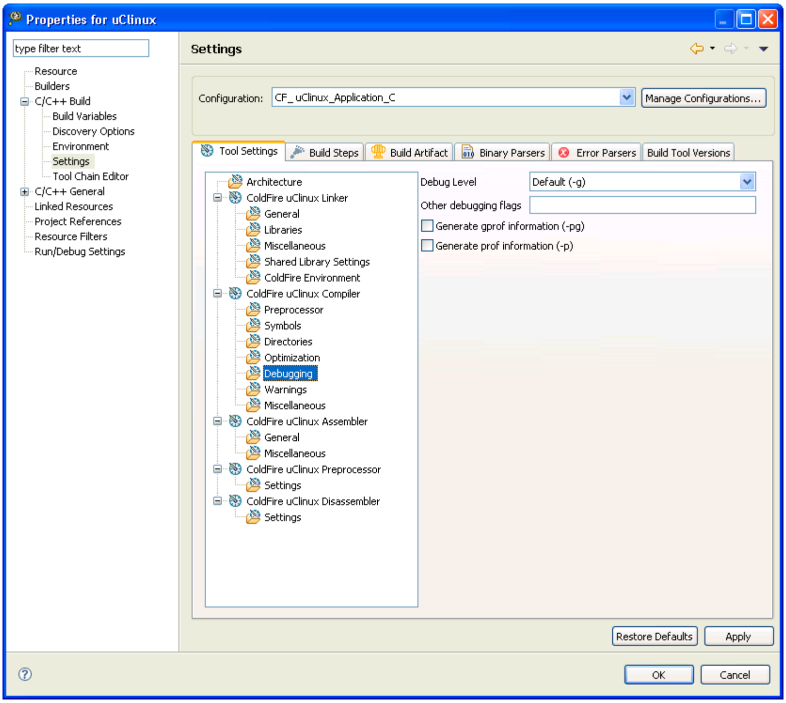
Option	Description
Optimization Level	Specify the optimizations that you want the compiler to apply to the generated object code: • **None (-O0) —** Disable optimizations. This setting is equivalent to specifying the `-O0` command-line option. The compiler generates unoptimized, linear assembly-language code. • **Optimize (-O1) —** The compiler performs all target-independent (that is, non-parallelized) optimizations, such as function inlining. This setting is equivalent to specifying the `-O1` command-line option. The compiler omits all target-specific optimizations and generates linear assembly-language code. • **Optimize more (-O2) —**The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the `-O2` command-line option. The compiler outputs optimized, non-linear, parallelized assembly-language code. • **Optimize most (-O3) —**The compiler performs all the level **2** optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the `-O3` command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level **-O2** optimizations.
Other optimization flags	Specify other optimization flags, like -Os (optimize for size) or specific optimizations if you would like to do a "fine-tuning" of optimizations.

ColdFire uClinux Compiler > Debugging

Use this panel to specify whether to generate symbolic information for debugging the build target.

Figure 8.15 shows the **Debugging** panel.

Figure 8.15 Tool Settings — ColdFire uClinux Compiler > Debugging



Configuration: CF_uClinux_Application_C

Debug Level: Default (-g)

Other debugging flags:

☐ Generate gprof information (-pg)

☐ Generate prof information (-p)

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Table 8.14 lists and describes the options for the **Debugging** panel.

Table 8.14 ColdFire uClinux Compiler > Debugging Options

Option	Description
Debug Level	Generates a compiled file containing debugging information. By default, it will be generated in DWARF-2 format.
Other debugging flags	Allows specifying other debugging options, like changing the format of debugging information or the level.

Table 8.14 ColdFire uClinux Compiler > Debugging Options (*continued*)

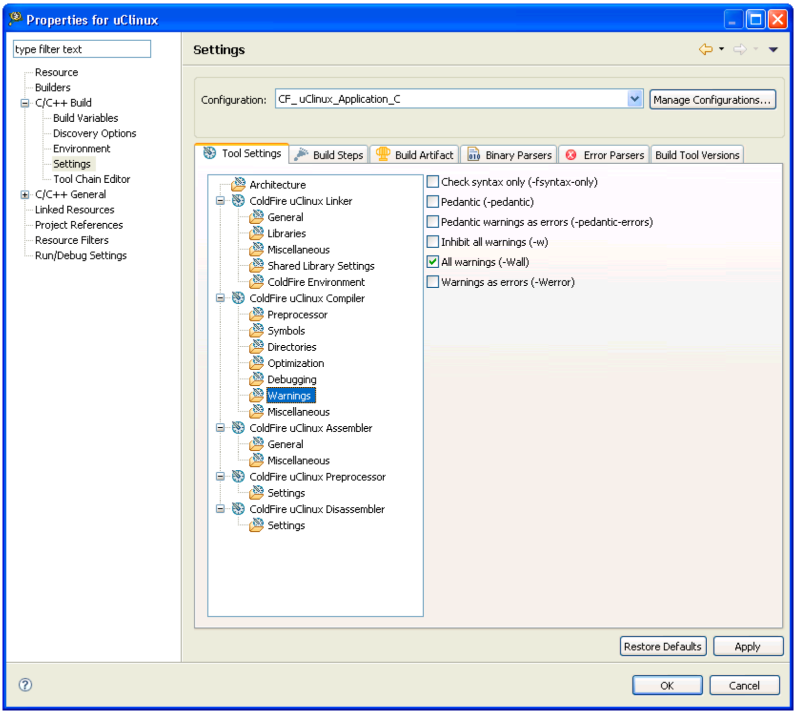
Option	Description
Generate gprof information (-pg)	Generate extra code to write profile information suitable for the analysis program `gprof`. You must use this option when compiling the source files you want data about, and you must also use it when linking.
Generate prof information (-p)	Generate extra code to write profile information suitable for the analysis program `prof`. You must use this option when compiling the source files you want data about, and you must also use it when linking.

ColdFire uClinux Compiler > Warnings

Use this panel to control how the ColdFire compiler formats the listing file, as well as the error and warning messages.

Figure 8.16 shows the **Warnings** panel.

Figure 8.16 Tool Settings — ColdFire uClinux Compiler > Warnings


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

☐ Check syntax only (-fsyntax-only)
☐ Pedantic (-pedantic)
☐ Pedantic warnings as errors (-pedantic-errors)
☐ Inhibit all warnings (-w)
☑ All warnings (-Wall)
☐ Warnings as errors (-Werror)

Table 8.15 lists and describes the options for the **Warnings** panel.

Table 8.15 ColdFire uClinux Compiler > Warnings

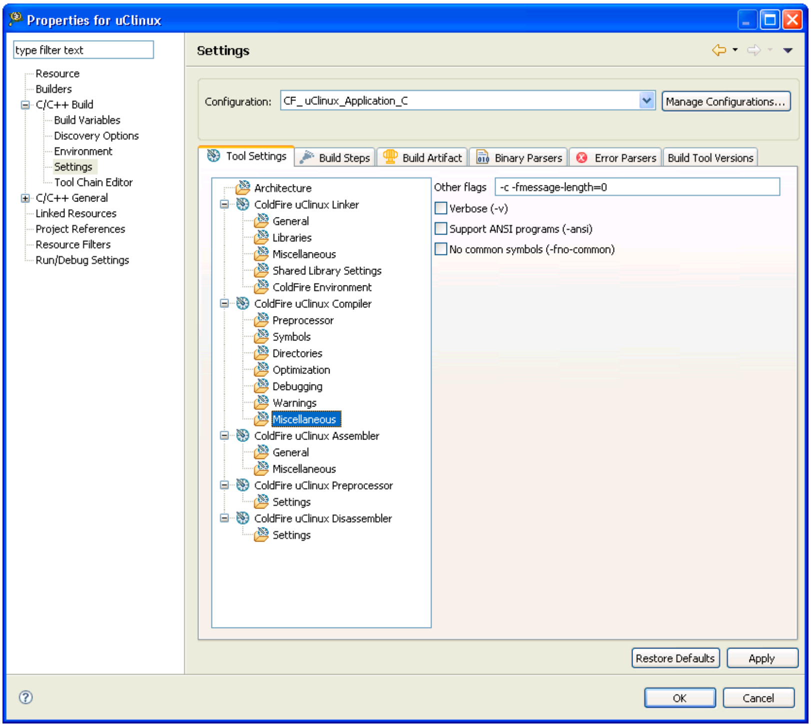
Option	Description
Check syntax only (-fsyntax-only)	Check the option if you want to scan the code only for syntax errors and nothing else. Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error. The following language-independent options do not enable specific warnings but control the kinds of diagnostics produced by GCC.
Pednatic (-pedantic)	Check is you want to issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any '-std' option used.
Pedantic warnings as errors (-pedantic-errors)	Check if you want warnings like `-pedantic`, except that errors are produced rather than warnings.
Inhibit all warnings (-w)	Check if you want to inhibit all warning messages.
All warnings (-Wall)	Check to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Warnings as errors (-Werror)	Check to make all warnings into hard errors. Source code which triggers warnings will be rejected.

ColdFire uClinux Compiler > Miscellaneous

Use this panel to specify miscellaneous settings for the **ColdFire uClinux Compiler**.

Figure 8.17 shows the **Miscellaneous** settings.

Figure 8.17 Tool Settings — ColdFire uClinux Compiler > Miscellaneous


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Other flags: -c -fmessage-length=0

☐ Verbose (-v)

☐ Support ANSI programs (-ansi)

☐ No common symbols (-fno-common)

Table 8.16 lists and describes the miscellaneous options for **ColdFire uClinux Linker**.

Table 8.16 ColdFire uClinux Compiler > Miscellaneous

Option	Description
Other flags	Specify additional command line options for the compiler; type in custom flags that are not otherwise available in the UI.
Verbose (-v)	Check to enable the verbose mode. Print out GNU CPP’s version number at the beginning of execution, and report the final form of the include path.

Table 8.16 ColdFire uClinux Compiler > Miscellaneous (*continued*)

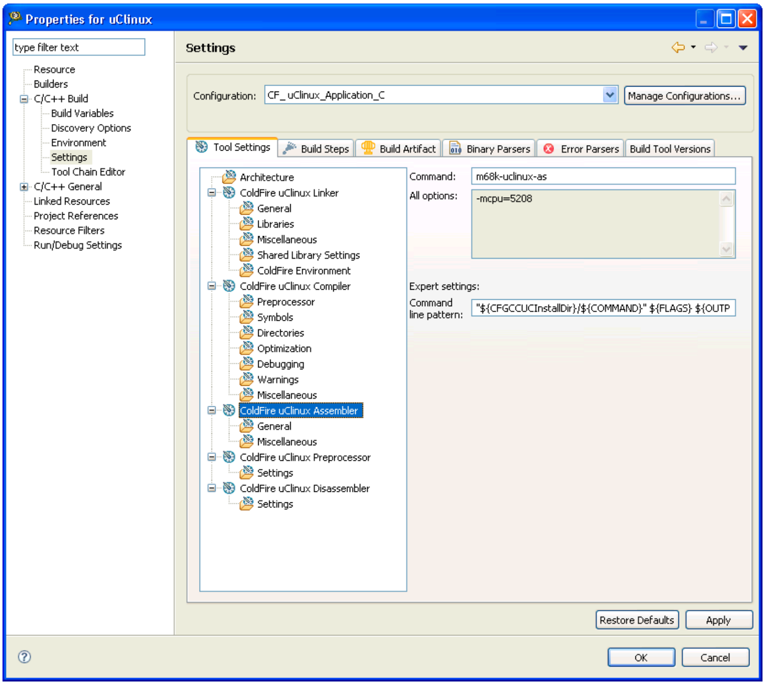
Option	Description
Support ANSI program (-ansi)	Check to specify the standard to which the code should conform.
No common symbols (-fno-common)	Check to control the placement of uninitialized global variables.

ColdFire uClinux Assembler

Use this panel to specify the command, options, and expert settings for the build tool assembler. Additionally, the Assembler tree control includes the general and include file search path settings.

Figure 8.18 shows the **Assembler** settings.

Figure 8.18 Tool Settings — ColdFire uClinux Assembler



Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Command: m68k-uclinux-as

All options: -mcpu=5208

Expert settings:

Command line pattern: "${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTP

Table 8.17 lists and defines each option of the **ColdFire uClinux Assembler** panel.

Table 8.17 ColdFire uClinux Assembler

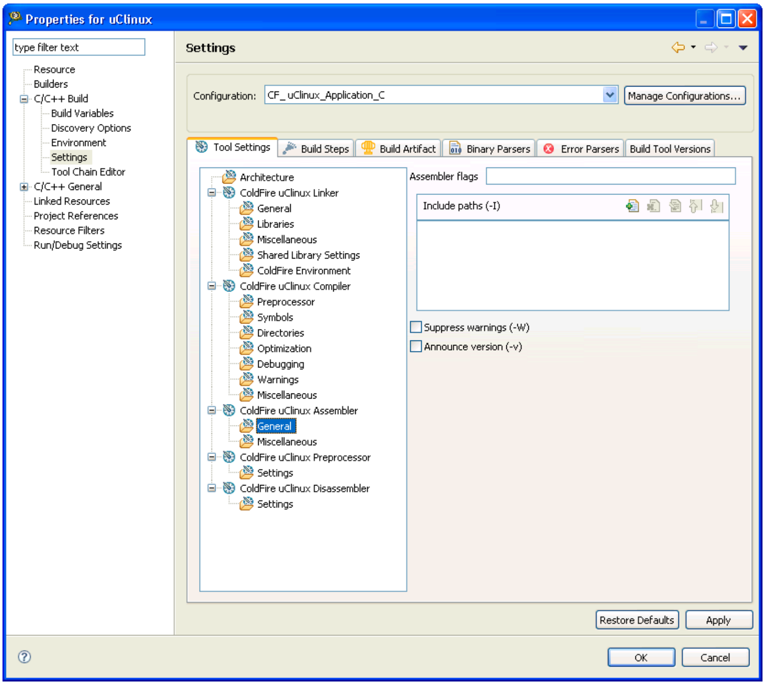
Option	Description
Command	Enter the command line arguments for the GCC assembler in the **Command Line Arguments** text box. The contents of this text box are passed as command-line switches in the gcc command line for each file in your project as they are assembled. Shows the location of the assembler executable file. Default value is: `m68k-uclinux-as`
All options	Shows the actual command line the assembler will be called with.
Expert Settings	
Command line pattern	Default value is: `"${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`

ColdFire uClinux Assembler > General

Use this panel to specify additional files the **ColdFire Assemble**r should use. You can specify multiple additional libraries and library search paths. Also, you can change the order in which the IDE uses or searches the libraries.

Figure 8.19 shows the **General** settings.

Figure 8.19 Tool Settings — ColdFire uClinux Assembler > General



Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

Assembler flags

Include paths (-I)

☐ Suppress warnings (-W)

☐ Announce version (-v)

Table 8.18 lists and defines each option of the **General** panel.

Table 8.18 ColdFire uClinux Assembler > General

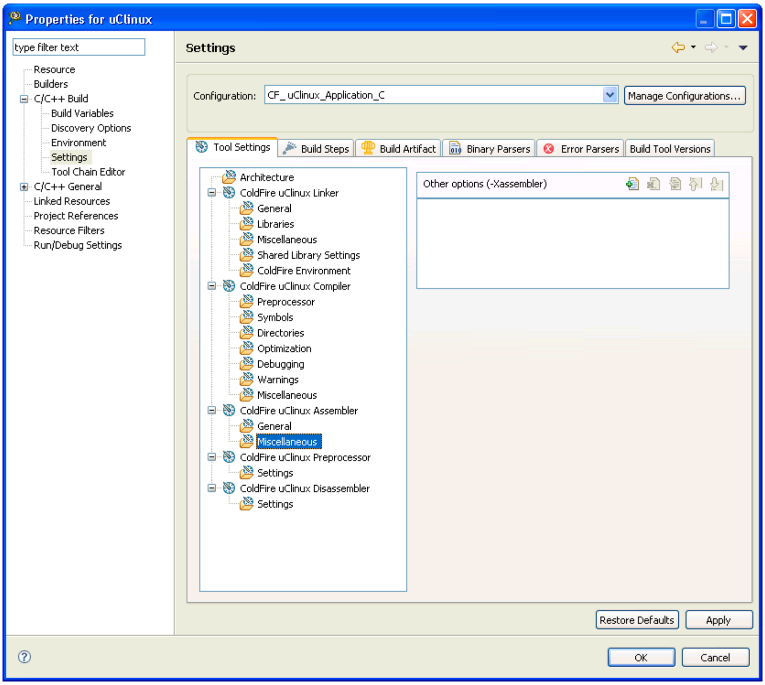
Option	Description
Assembler flags	Specify options that will be passed to the assembler.
Include paths (-I)	Use this option to add a path to the list of directories as searches for files specified in `.include` directives.
Suppress warnings (-W)	Check to inhibit all warning messages.
Announce version (-v)	Check to find out what version of assembler is running.

ColdFire uClinux Assembler > Miscellaneous

Use this panel to specify miscellaneous settings for the **ColdFire uClinux Assembler**.

Figure 8.20 shows the **Miscellaneous** settings.

Figure 8.20 Tool Settings — ColdFire uClinux Assembler > Miscellaneous



Configuration: CF_ uClinux_Application_C

Other options (-Xassembler)

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Table 8.19 lists and defines each option of the **Miscellaneous** panel.

Table 8.19 ColdFire uClinux Assembler > Miscellaneous

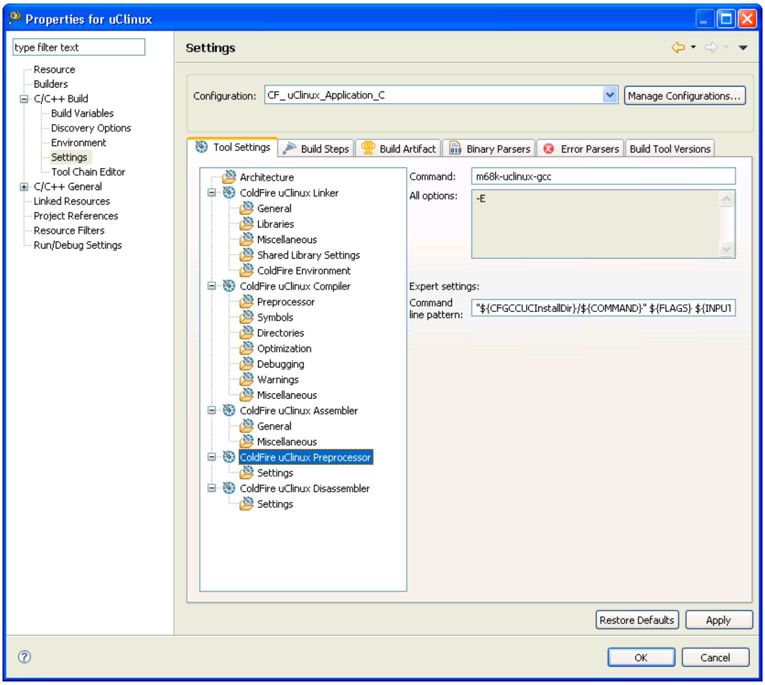
Option	Description
Other options (-Xassembler)	Provide an option to the assembler. You can use this to supply system-specific assembler options which GCC does not know how to recognize. If you want to pass an option that takes an argument, you must use `-Xassembler` twice, once for the option and once for the argument.

ColdFire uClinux Preprocessor

Use this panel to specify preprocessor behavior and define macros.

Figure 8.21 shows the **Preprocessor** panel.

Figure 8.21 Tool Settings — ColdFire uClinux Preprocessor



Configuration: CF_uClinux_Application_C

Command: m68k-uclinux-gcc

All options: -E

Expert settings:

Command line pattern: "${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${INPUT

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Table 8.20 lists and defines each option of the **Preprocessor** panel.

Table 8.20 ColdFire uClinux Preprocessor

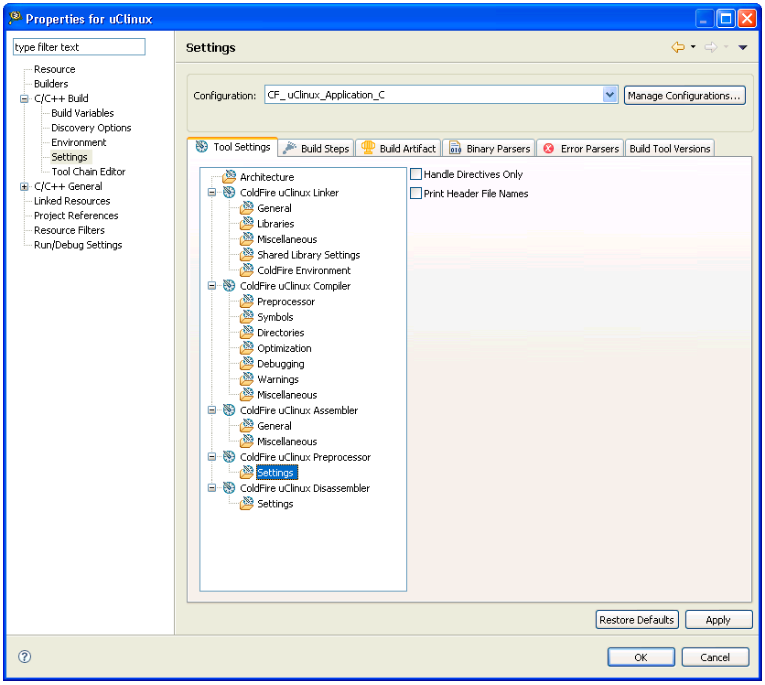
Option	Description
Command	Provide the command line arguments for the GCC assembler in the **Command Line Arguments** text box. The contents of this text box are passed as command-line switches in the gcc command line for each file in your project as they are assembled. Shows the location of the linker executable file. Default value is: `m68k-uclinux-gcc`
All options	Shows the actual command line the preprocessor will be called with.
Expert Settings	
Command line pattern	Default value is: `"${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${INPUTS}`

ColdFire uClinux Preprocessor > Settings

Use this panel to specify preprocessor behavior.

Figure 8.22 shows the **Preprocessor** panel.

Figure 8.22 Tool Settings — ColdFire uClinux Preprocessor > Settings



Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

☐ Handle Directives Only

☐ Print Header File Names

Table 8.21 lists and defines each option of the **Settings** panel.

Table 8.21 ColdFire uClinux Preprocessor > Settings

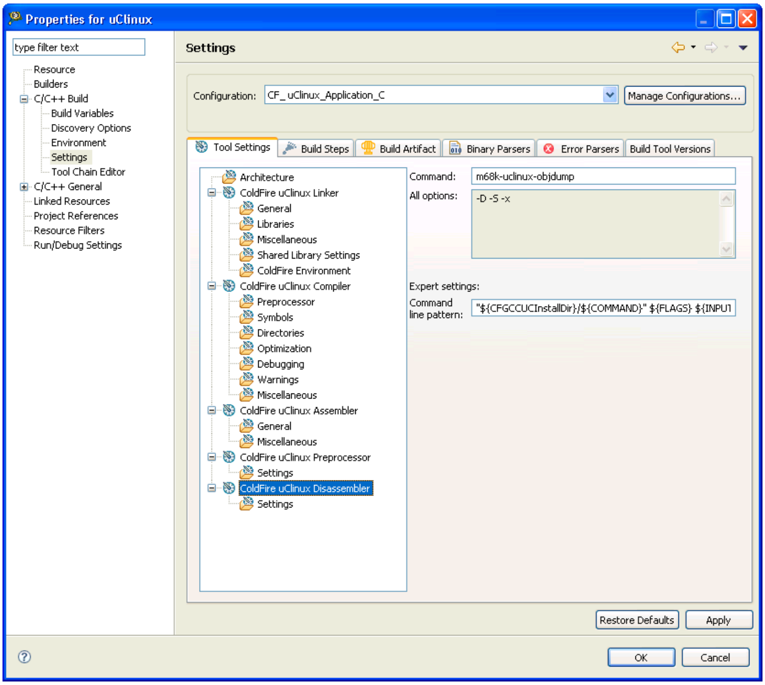
Option	Description
Handle Directives Only	Select to handle directives only.
Print Header File Names	Select if you want to print header filenames.

ColdFire uClinux Disassembler

Use this panel to specify the command, options, and expert settings for the **ColdFire uClinux Disassembler**.

Figure 8.23 shows the **ColdFire Disassembler** page.

Figure 8.23 Tool Settings — ColdFire uClinux Disassembler


Configuration: CF_ uClinux_Application_C

Command: m68k-uclinux-objdump

All options: -D -S -x

Expert settings:

Command line pattern: "${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${INPU1

Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Table 8.22 lists and defines each option of the **Settings** panel.

Table 8.22 ColdFire uClinux Disassembler

Option	Description
Command	You can enter the command line arguments for the GCC disassembler in the **Command** text box. The contents of this text box are passed as command-line switches in the gcc command line for each file in your project as they are assembled. Shows the location of the linker executable file. Default value is: `m68k-uclinux-objdump`
All options	Shows the actual command line the preprocessor will be called with.

Table 8.22 ColdFire uClinux Disassembler (*continued*)

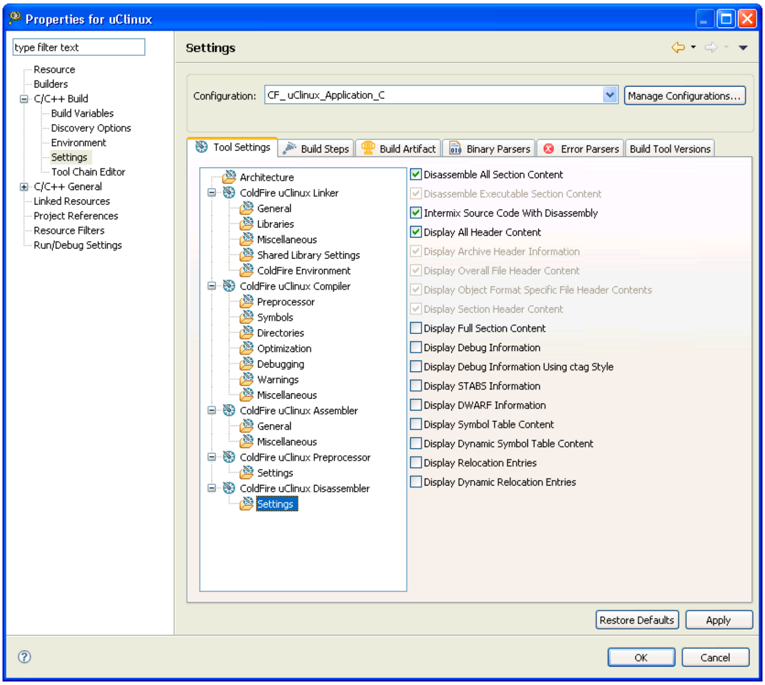
Option	Description
Expert Settings	
Command line pattern	Default value is: `"${CFGCCUCInstallDir}/${COMMAND}" ${FLAGS} ${INPUTS}`

ColdFire uClinux Disassembler > Settings

Use this panel to control how the disassembler formats the listing file, as well as error and warning messages. You can specify verbosity of messages, whether to show headers, core modules, extended mnemonics, addresses, object or source code, data modules, exception tables, and debug information.

Figure 8.24 shows the **ColdFire uClinux Disassembler** settings.

Figure 8.24 Tool Settings — ColdFire uClinux Disassembler


Properties for uClinux
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire uClinux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire uClinux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire uClinux Assembler
General
Miscellaneous
ColdFire uClinux Preprocessor
Settings
ColdFire uClinux Disassembler
Settings
Restore Defaults
Apply
OK
Cancel

Configuration: CF_uClinux_Application_C

- ☑ Disassemble All Section Content
- ☑ Disassemble Executable Section Content
- ☑ Intermix Source Code With Disassembly
- ☑ Display All Header Content
- ☑ Display Archive Header Information
- ☑ Display Overall File Header Content
- ☑ Display Object Format Specific File Header Contents
- ☑ Display Section Header Content
- ☐ Display Full Section Content
- ☐ Display Debug Information
- ☐ Display Debug Information Using ctag Style
- ☐ Display STABS Information
- ☐ Display DWARF Information
- ☐ Display Symbol Table Content
- ☐ Display Dynamic Symbol Table Content
- ☐ Display Relocation Entries
- ☐ Display Dynamic Relocation Entries

Table 8.23 lists and defines each option of the **Settings** panel.

Table 8.23 ColdFire uClinux Disassembler

Option	Description
Disassemble All Section Content	Disassembles all section content and sends the output to a file. This command is global and case-sensitive.
Disassemble Executable Section Content	Disassembles all executable content and send output to a file.
Intermix Source Code with Disassembly	Turn jbsr into jsr.
Display All Header Content	Display the contents of all headers.
Display Archive Header Information	Display archive header information.
Display Overall File Header Content	Display the contents of the overall file header.
Display Full Section Content	Display the full section of the file.
Display Debug Information	Display debug information in the object file.
Display Debug Information Using ctag Style	Display debug information using the ctags style.
Display STABS Information	Displays any STABS information in the file, in raw form.
Display DWARF Information	Displays any DWARF information in the file.
Display Symbol Table Content	Displays the contents of the symbol tables.
Display Dynamic Symbol Table Content	Displays the contents of the dynamic symbol table.
Display Relocation Entries	Displays the relocation entries in the file.
Display Dynamic Relocation Entries	Displays the dynamic relocation entries in the file.

9

Debugging Embedded Linux

This chapter explains how to use the CodeWarrior tools to debug embedded Linux® programs on ColdFire® hardware.

NOTE The chapter covers those aspects of debugging that are specific to the ColdFire platform. Refer to the *Freescale Eclipse Extensions Guide* for debugger information that applies to all CodeWarrior products.

The topics in this chapter are:

- Debugging Linux Application
  - Working With CodeWarrior TRK
  - Viewing Multiple Processes and Threads
  - Debugging Applications that use fork() and exec() System Calls
- Debugging Shared Library
- Debugging U-Boot Loader
- Debugging Linux Kernel

Debugging Linux Application

This sections shows you how to use the CodeWarrior Target Resident Kernel (TRK) to debug a Linux application.

Working With CodeWarrior TRK

This section describes CodeWarrior TRK and provides information related to using it with CodeWarrior projects.

For embedded Linux development, CodeWarrior TRK is a user-level application that resides on target embedded Linux systems and accepts connections from the CodeWarrior debugger. You use the CodeWarrior remote connections feature to download and debug applications built with CodeWarrior projects. The CodeWarrior debugger connects to CodeWarrior TRK on the remote target system through a serial or Ethernet connection.

On embedded Linux systems, a regular Linux application, named `AppTRK` runs on the remote target system along side the program you are debugging to provide application-level debug services to the CodeWarrior debugger.

To use CodeWarrior TRK to debug a Linux application, follow this high-level process:

- Install CodeWarrior TRK on Target System
- Start CodeWarrior TRK on Target System
- Creating Linux/uClinux Application Project
- Debug Linux Application

Install CodeWarrior TRK on Target System

To connect the CodeWarrior debugger to AppTRK, the AppTRK binary executable file must be installed and running on the remote target system. Once AppTRK is running on the target system, the debugger can upload your application and debug the application on the target system.

NOTE If AppTRK is not present on a given target system, you must use File Transfer Protocol (FTP) or another file transfer facility to download the CodeWarrior TRK binary executable file, `AppTRK.elf`, to a suitable location on the file system of the target system. The AppTRK executable is `AppTRK.elf` only on MMU-enabled boards. For MMU-less boards it is `AppTRK.flt`.

Start CodeWarrior TRK on Target System

How you start AppTRK on the target hardware depends on the type of connection between the host computer and that target hardware:

- Transmission Control Protocol/Internet Protocol (TCP/IP Connection) — Host computer communicates with the target hardware over a TCP/IP connection.
- Serial Connection — Serial cable connects the host computer to the target hardware.

Follow the procedure for starting AppTRK that matches your connection type.

TCP/IP Connection

Follow these steps to start AppTRK through a TCP/IP connection:

1. Connect a serial cable between the host computer’s serial port and the second serial port of the target system.
2. On the host computer, start a terminal emulation program (such as minicom).
3. Configure the terminal emulation program with baud rate, stop bit, parity, and handshake settings appropriate for the target system.

4. Connect the terminal emulator to the target system.

   A command prompt appears in the terminal emulation program.

5. Boot the system. Log in as the root user.
6. Connect to the remote target system.
   a. On the host computer, open a new terminal window.
   b. At the command prompt in the terminal window, enter this command, where *IPAddress* represents the target system's IP address:

      ```
      telnet IPAddress
      ```

      The telnet client connects to the telnet daemon on the target system.

7. Navigate to the directory that contains the `AppTRK.elf` binary executable file.

   The system changes the current working directory.

8. Specify the command (where *Port* is the listening port number optionally specified in the **Connections** panel of Debug window— typically `1000`):

   ```
   ./AppTRK.elf :Port
   ```

   CodeWarrior TRK starts on the target system, and listens to the specified TCP/IP port for connections from the CodeWarrior IDE.

TIP To continue use of the terminal session after launching AppTRK, start AppTRK as a background process by appending the ampersand symbol (&) to the launch command. For example, to start AppTRK as a background process listening to TCP/IP port number 6969, enter the command: `./AppTRK.elf :6969 &`.

Serial Connection

Follow these steps to launch AppTRK through a serial connection:

TIP To improve your debugging experience, we recommend the host computer running the IDE have two serial ports. In an ideal scenario, you would connect one serial port of the host computer to the first serial port of the target board to monitor startup and console log messages. You would then connect another serial port of the host computer to the second serial port of the target board; the debugger would use this connection to communicate with AppTRK on the target system.

1. Connect a serial cable between the host computer's serial port and the second serial port of the target system.
2. On the host computer, start a terminal emulation program (such as minicom).
3. Configure the terminal emulation program with baud rate, stop bit, parity, and handshake settings appropriate for the target system.

4. Connect the terminal emulator to the target system.

   A command prompt appears in the terminal emulation program.

5. Boot the system. Log in as the root user.
6. Use the `cd` command at the command prompt to navigate to the directory where the AppTRK binary executable file, `AppTRK.elf`, resides on the target system.

   The system changes the current working directory.

7. Configure the serial port on which CodeWarrior TRK is to connect.
   a. Enter this command:

      ```
      stty -F /dev/ttyS1 raw
      ```

      This command configures the serial port for raw mode of operation. If you do not use raw mode, special characters sent as part of packets may be interpreted (dropped), causing the connection to break.

   b. Enter this command:

      ```
      stty -F /dev/ttyS1 ispeed 115200
      ```

      The serial input speed is set to 115200 baud.

   c. Enter this command:

      ```
      stty -F /dev/ttyS1 ospeed 115200
      ```

      The serial output speed is set to 115200 baud.

   d. Enter this command:

      ```
      stty -F /dev/ttyS1 crtscts
      ```

      The terminal emulation program enables handshake mode

   e. Enter this command:

      ```
      stty -a -F /dev/ttyS1
      ```

      The system displays the current device settings.

8. Enter the command:

   ```
   ./AppTRK.elf /dev/ttyS1
   ```

   AppTRK launches on the remote target system.

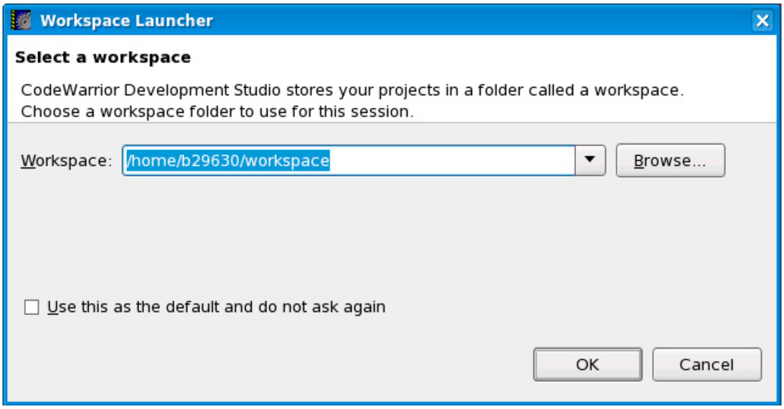
Creating Linux/uClinux Application Project

To create a Linux/uClinux Application Project using the **New Linux/uClinux Application Project** wizard, perform these steps.

1. Open a new terminal window and change the working directory to <*`CWInstall`*>*`/eclipse/`*, where <*`CWInstall`*> is the directory in which you installed the CodeWarrior software.
2. Issue the command: `./cwide`.

   The IDE launches and the **WorkSpace Launcher** dialog box prompts you to select a workspace to use.

Figure 9.1 WorkSpace Launcher Dialog Box



Select a workspace

CodeWarrior Development Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

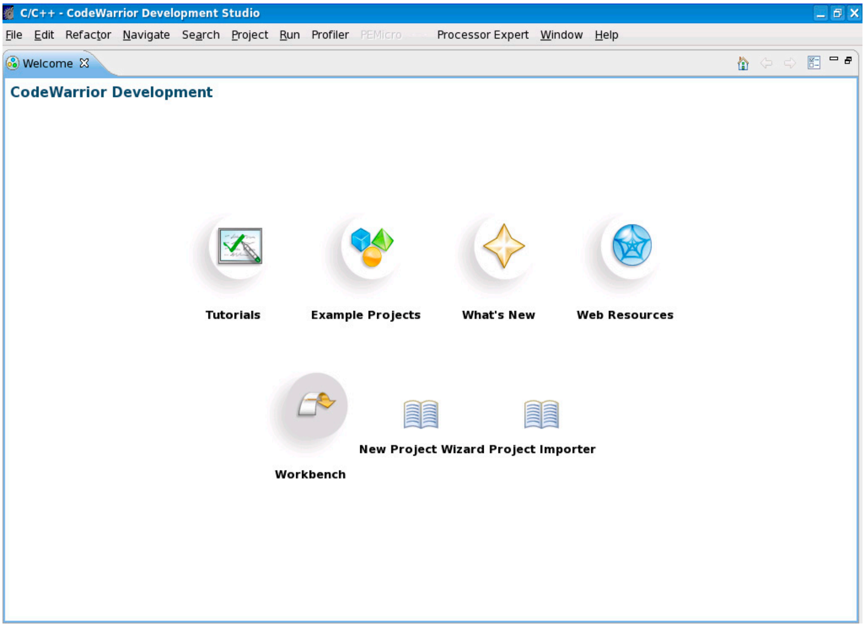
Workspace: /home/b29630/workspace

3. Click **OK** to accept the default workspace. To use a workspace different from the default, click **Browse** and specify the desired workspace.

   The IDE starts and displays the **Welcome** page.

NOTE You can also select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.

Figure 9.2 Welcome Page


C/C++ - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Run Profiler PEMicro Processor Expert Window Help
Welcome
CodeWarrior Development
Tutorials
Example Projects
What's New
Web Resources
Workbench
New Project Wizard
Project Importer

4. Click the **Go to Workbench** link.

   The **Workbench** window opens.

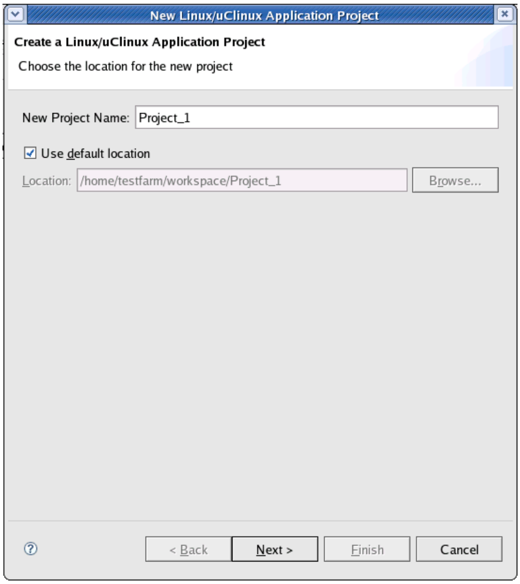
5. Select **File > New > Linux/uClinux Application Project**, from the IDE menu bar.

   The **Create a Linux/uClinux Application Project** page of the **New Linux/uClinux Application Project** wizard appears.

6. Specify a name for the new project. For example, enter the project name as `Project_1`.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify different location for the new project. By default, the **Use default location** checkbox is checked.

Figure 9.3 Create a Linux/uClinux Application Project Page



New Linux/uClinux Application Project

Create a Linux/uClinux Application Project

Choose the location for the new project

New Project Name: Project_1

☑ Use default location

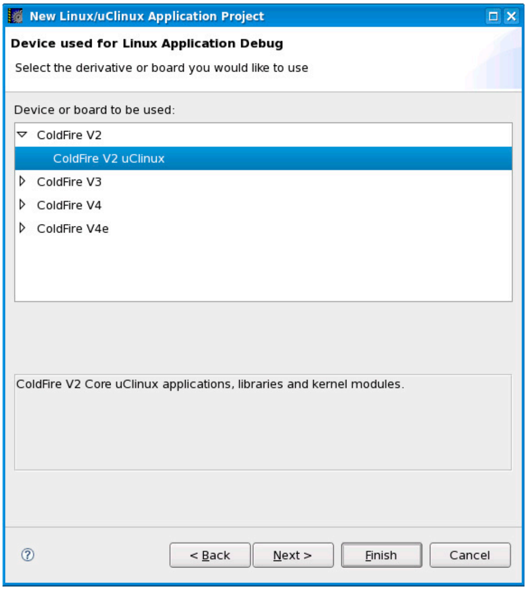
Location: /home/testfarm/workspace/Project_1

7. Click **Next**.

   The **Device used for Linux Application Debug** page appears.

8. Expand the tree control and select the derivative or board you would like to use. For example, select `ColdFire V2 > ColdFire V2 uClinux.`

Figure 9.4 Device used for Linux Application Debug Page

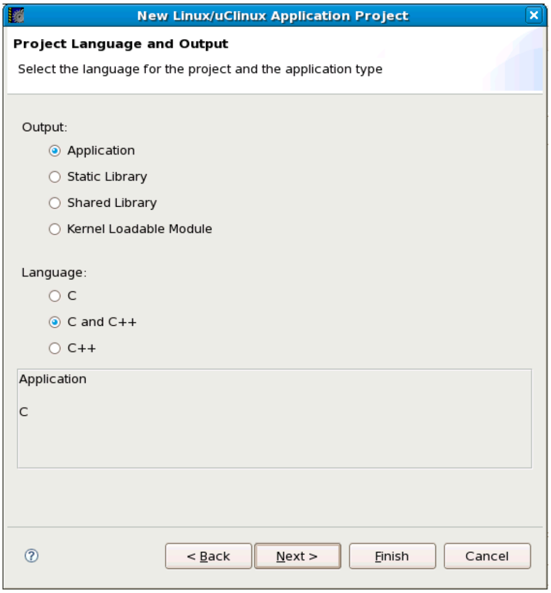

New Linux/uClinux Application Project
Device used for Linux Application Debug
Select the derivative or board you would like to use
Device or board to be used:
ColdFire V2
ColdFire V2 uClinux
ColdFire V3
ColdFire V4
ColdFire V4e
< Back
Next >
Finish
Cancel

ColdFire V2 Core uClinux applications, libraries and kernel modules.

9. Click **Next**.

   The **ProjectLanguage and Output** page (Figure 9.5) appears.

Figure 9.5 Project Language and Output Page



New Linux/uClinux Application Project

Project Language and Output

Select the language for the project and the application type

Output:

- ● Application
- ○ Static Library
- ○ Shared Library
- ○ Kernel Loadable Module

Language:

- ○ C
- ● C and C++
- ○ C++

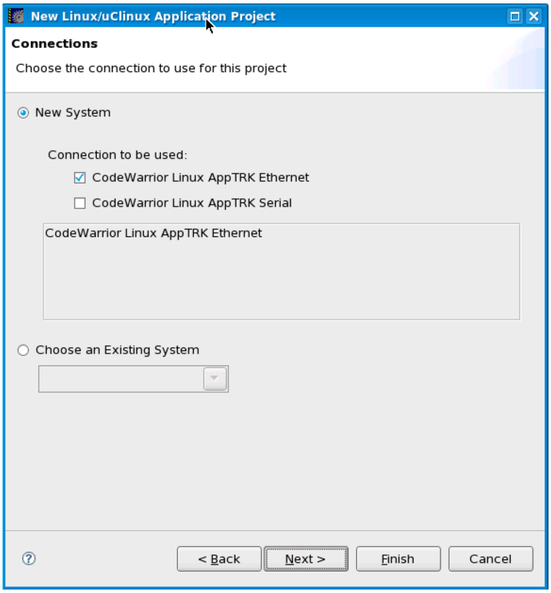
Application

C

10. Select the output type and the programming language you want to use for this project. For example, select **Application** and **C and C++**.
11. Click **Next**.

    The **Connections** page appears.

Figure 9.6 Connections Page

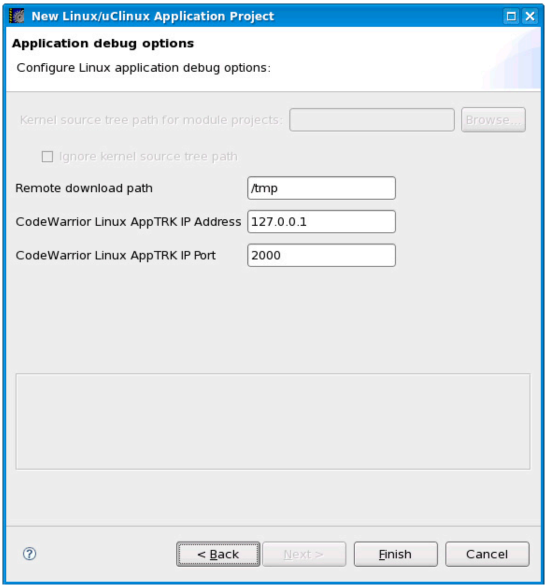


New Linux/uClinux Application Project
Connections
Choose the connection to use for this project
● New System
Connection to be used:
☑ CodeWarrior Linux AppTRK Ethernet
☐ CodeWarrior Linux AppTRK Serial
CodeWarrior Linux AppTRK Ethernet
○ Choose an Existing System
< Back
Next >
Finish
Cancel

12. Check the appropriate connection.
13. Click **Next**.

    The **Application debug options** page appears.

Figure 9.7 Application Debug Options Page



New Linux/uClinux Application Project
Application debug options
Configure Linux application debug options:
Kernel source tree path for module projects:
Browse...
☐ Ignore kernel source tree path

Remote download path: /tmp

CodeWarrior Linux AppTRK IP Address: 127.0.0.1

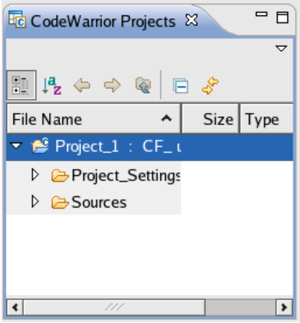
CodeWarrior Linux AppTRK IP Port: 2000

< Back
Next >
Finish
Cancel

14. From the list, select the method with which you want the IDE to connect to the target system.
15. In the **Remote download path** text box, specify the path. By default, it is `/tmp`.
16. In the **CodeWarrior TRK IP Address** and **CodeWarrior IP Port** text boxes, enter the IP address and listening port of the target system. By default, the IP address is `127.0.0.1` and the port number is `2000`.
17. Click **Finish**.

    The wizard closes. The IDE generates a new project according to your specifications. The **CodeWarrior Projects** view (Figure 9.8) appears.

Figure 9.8 CodeWarrior Projects View


CodeWarrior Projects
File Name
Size
Type
Project_1 : CF_
Project_Settings
Sources

18. Select **Run** > **Debug Configurations**.

    The **Debug Configurations** dialog box appears.

19. Select **CodeWarrior Download** on the left-hand side of the **Debug Configurations** dialog box.

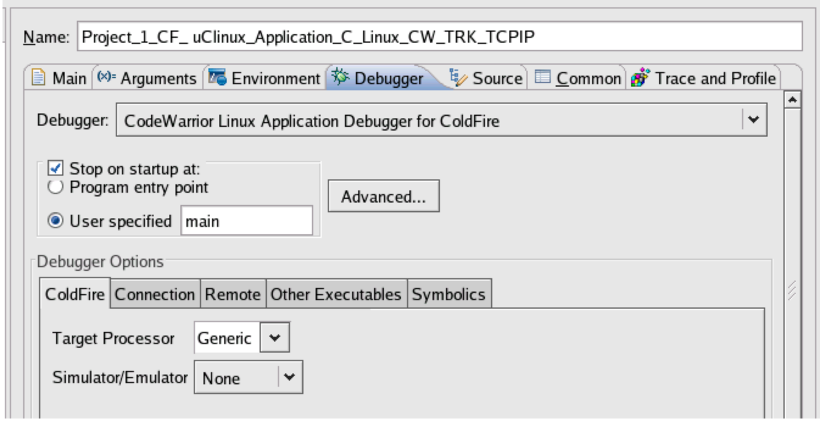
20. Click the **Debugger** tab.

    The debugger setting page appears in the area beneath the tabs.

21. Select **CodeWarrior Linux Application Debugger for ColdFire** from the **Debugger** drop-down list.

    The corresponding set of sub-pages appears in the **Debug Configurations** dialog box as in Figure 9.9.

Figure 9.9 Debug Window — Linux Applications


Name: Project_1_CF_ uClinux_Application_C_Linux_CW_TRK_TCPIP

Debugger: CodeWarrior Linux Application Debugger for ColdFire

☑ Stop on startup at:
○ Program entry point
● User specified main

Target Processor: Generic

Simulator/Emulator: None

Main
Arguments
Environment
Debugger
Source
Common
Trace and Profile
Advanced...
Debugger Options
ColdFire
Connection
Remote
Other Executables
Symbolics

22. If required, specify a function or address in the application where you want the program control to stop first in the debug session:

   a. Check the **Stop on startup** at checkbox.

      The IDE enables the corresponding text box.

   b. Enter in the text box an address or a function inside the application.

23. Click the **ColdFire** tab.

    The **ColdFire** settings appear.

24. From the **Target Processor** drop-down list, select Generic.
25. Click the **Connection** tab.

    The **Connection** tab settings appear.

26. Configure the **Connection** sub-page settings as appropriate for the connection between the host computer and the target hardware.
27. Click the **Remote** tab.

    The **Remote** tab settings appear.

28. In the Remote download path text box, specify the path of a host-computer directory to which the Linux application running on the target hardware can read and write files.
29. If required, specify information about other executable files to debug in addition to the Linux application:

    a. Click the **Other Executables** tab.

       The **Other Executables** tab settings appear.

    b. Use the **Add, Change, or Remove** buttons to specify information about other executable files.

    c. Click the **Apply** button.

    The IDE saves the pending changes you made to the launch configuration.

You just finished creating a CodeWarrior Download launch configuration that you can use to debug the Linux application.

Debug Linux Application

After you create a CodeWarrior Download launch configuration for debugging the Linux application on the target hardware, you can use that launch configuration to start a debugging session.

Follow these steps to start a debugging session that you can use to debug the Linux application:

1. On the left-hand side of the **Debug Configurations** dialog box, select the CodeWarrior Download launch configuration that you created to debug the Linux application.
2. Click the **Debug** button.

   The IDE uses the selected CodeWarrior Download launch configuration to start a debugging session and brings up the Debug perspective.

Viewing Multiple Processes and Threads

When you debug an application, the CodeWarrior debugger opens the Debug perspective. In this perspective, you can view only processes and threads/tasks on which debugger is attached.

For Linux debugging, you can view all processes on target in the **System Browser** view.

To view processes and threads in **System Browser** view:

1. Open a Linux application in the CodeWarrior IDE.
2. Select **Run > Debug**.

   The **Debug** perspective appears.
3. While the application is running, select **Window > Show View > Other**.

   The **Show View** dialog box appears.
4. Expand the **Debug** group and select **System Browser**.
5. Click **OK**.

   The **System Browser** view appears with the process and the thread information.

Debugging Applications that use fork() and exec() System Calls

The CodeWarrior debugger lets you debug a program that contains `fork()` and `exec()` system calls. Table 9.1 summarizes the descriptions of these system calls.

Table 9.1 fork() and exec() Description

System Call	Description
`fork()`	The `fork()` system call is used as a generic call on Linux systems to create a new process. The `fork()` call creates a new process, which is the exact replica of the process that creates it. The only difference is in the Process ID (PID) returned by the fork system call. The value of PID returned in the parent process is the PID of the child, whereas in the child process the PID value returned is zero.
`exec()`	The `exec()` system call launches a new executable in an already running process. The debugger destroys the instance of the previous executable loaded into that address space and a new instance is created.

NOTE You can also pick up sample applications from the `<CWInstallDir>\MCU\ColdFire Tools\Linux_Examples\CF\` folder, where

For debugging applications that use the `fork()` system call, the `fork()` system call is overridden by the `clone()` system call. The `clone()` system call is called with the flag `CLONE_PTRACE` instead of the `fork()` system call. Calling the `clone()` system call with the flag `CLONE_PTRACE` causes:

- The operating system to attach CodeWarrior TRK to the child process.
- The child process to stop with a `SIGTRAP` on return from the `clone()` system call.

To call the `clone()` system call transparently while debugging programs that contain the `fork()` system call, you need to add a static library to your project. The source code for building the static library is described later in this section.

NOTE The static library necessary for debugging programs that contain the `fork()` system call must be added to the project. The location is:
`CWInstallDir\MCU\ColdFire Tools\Linux_Examples\`,
where

Before you start debugging an application ensure that you have:

- created a IP connection between the host computer and the remote target and
- launched AppTRK on the remote target.

The following steps demonstrate how to use the CodeWarrior IDE to debug programs that contain `fork()` and `exec()` system calls:

1. As a first step, create a project with the following settings (Table 9.2).

Table 9.2 Static Library Project Settings

Project Name:	Fork
Location of the Project	`<CWInstallDir>\ColdFire Tools\Linux_Examples\CF\, where`
Languages	C
Stationery Type	Linux Application

   The IDE creates a project with a debug launch configuration.

2. Rename the default debug configuration to **Fork**.
3. Create a new build configuration. Right-click on the project folder and select **Build Configurations** > **Manage**.

   The **Fork: Manage Configurations** dialog box appears.

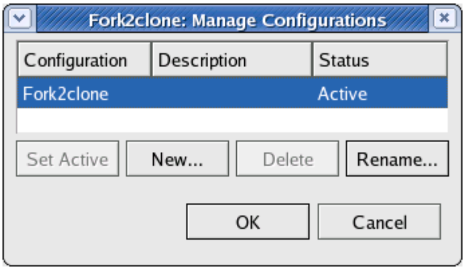
4. Click **New** to create a new build configuration.

   The **Create New Configuration** dialog box appears.

5. In the **Name** field, enter the configuration name as **Fork2clone**.
6. Select the **Existing configuration** option in the **Copy settings from** group.
7. Click **OK**.

   The **Fork: Manage Configurations** dialog box appears.

Figure 9.10 Fork: Manage configurations Dialog Box



8. Set the `Fork2clone` build configuration as active.
9. To build the `Fork2clone` configuration, right-click and select **Build Project** from the context menu.

   The CodeWarrior IDE builds the project and stores the support library, `libfork2clone.a` in the *Output* directory within the project directory.

NOTE Remember to build the `Fork2clone` build configuration before the `Fork` build configuration to avoid getting a library file missing error as the `libfork2clone.a` is used in the Fork project.

10. Specify the linker settings and add the support library to the project.
    a. Right-click the `Fork` build configuration in the **CodeWarrior Projects** view.
    b. Select **Properties** from the context menu.

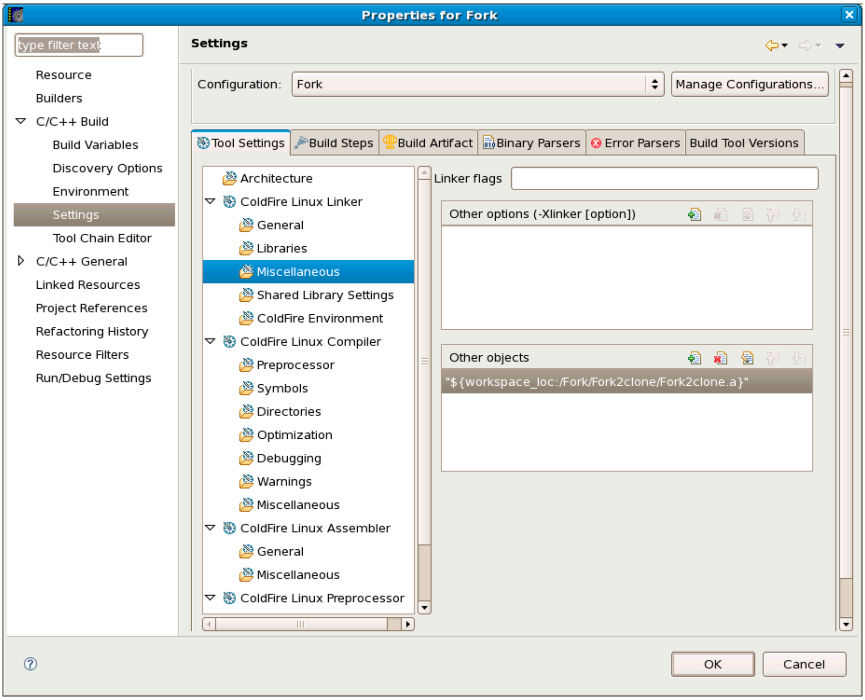
       The **Properties** window for the shared library project appears.
    c. Select C/C++ Build > Settings.

       The **Settings** page of the **Properties for <*project*>** dialog box appears.
    d. Select **ColdFire Linux Linker > Miscellaneous**, in the **Tool Settings** page.
    e. In the **Other objects** panel click the **Add** icon.
    f. Browse to the newly built library, `fork2clone.a`.

       The settings appear as seen in Figure 9.11.

NOTE These settings enable the CodeWarrior IDE linker to locate the static library `libfork2clone.a`. For detailed information on other linker command line arguments, refer GNU linker manuals.

Figure 9.11 Miscellaneous Linker Settings — Fork Project



Properties for Fork
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Linked Resources
Project References
Refactoring History
Resource Filters
Run/Debug Settings
Settings
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Build Tool Versions
Architecture
ColdFire Linux Linker
General
Libraries
Miscellaneous
Shared Library Settings
ColdFire Environment
ColdFire Linux Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ColdFire Linux Assembler
General
Miscellaneous
ColdFire Linux Preprocessor
Linker flags
Other options (-Xlinker [option])
Other objects
OK
Cancel

Configuration: Fork

Other objects: "${workspace_loc:/Fork/Fork2clone/Fork2clone.a}"

11. Click **OK** to close the **Properties for *<project>*** window.
12. Remove the default `main.c` file from the project.
13. Add a new `db_fork.c` file to the project.
14. Enter the source code shown in Listing 9.1 into the editor window of `db_fork.c` file.

Listing 9.1 Source Code for db_fork.c

```
/*
  --------------------------
  User Include files
  --------------------------
*/

#include "db_fork.h"

/*
  -------------------------
  Main Program
  -------------------------
*/
int __libc_fork(void)
{
  return( __db_fork() );
}
extern __typeof (__libc_fork) __fork __attribute__ ((weak, alias
("__libc_fork")));
extern __typeof (__libc_fork) fork __attribute__ ((weak, alias
("__libc_fork")));
```

15. Create a header file `db_fork.h` in your project directory and add the code shown in Listing 9.2 into the header file.

Listing 9.2 Source Code for db_fork.h

```
#include <asm/unistd.h>
#include <sys/syscall.h>
#include <errno.h>
#include <signal.h>
#include <sched.h>
#define __NR___db_clone__NR_clone
#define __db_fork()
syscall(__NR___db_clone, SIGCHLD | CLONE_PTRACE, 0);
```

16. Create another project; `Exec` and create two new build configurations with the following settings (Table 9.3):

Table 9.3 Fork and Exec Example Project Settings

Project Name	Exec
Location of the Project	Fork&Exec workspace
Languages	C
Output Type	Application
Build Configurations	- Parent_debug - Child_debug

17. Add the source files `exec.c and exec-1.c` to the `Exec` project.

- `exec.c` — the code of the parent process

- `exec-1.c` — generates the executable file `exec-1.elf`

The code of the parent process creates a forked process (child process) when the `__db_fork` function executes. The debugger opens a separate thread window for the child process. When the child process finishes executing, the debugger closes the thread window. To debug the code of the child process, you need to set a breakpoint in the child process code. You can debug the code of the child process the same way you debug code of any other process.

The code of both child and parent processes contain `exec()` function calls that execute the `exec-1.elf`.

As you step through the code of the child process, the `exec()` function call executes and a separate debugger window for the `exec-1.elf` appears. You can perform normal debug operations in this window. Similarly, you step through the code of the parent process to execute the `exec()` system call. The debugger destroys the instance of the previous file (`exec.elf`) and creates a new instance for the `exec-1.elf` file.

18. Enter the source code shown in Listing 9.3 into the editor window of `fork.c` file.

Listing 9.3 Source Code for fork.c

```
/*
/*
 *   fork.c
 *
 */

/*------------------------------------------------------------------*
 System Include files
 *------------------------------------------------------------------*/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/ptrace.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <signal.h>
#include <sched.h>
#include <fcntl.h>
#include <dlfcn.h>

/*-------------------------------------------------------------------*
  Function Prototypes
 * ------------------------------------------------------------------*/
int fn1(int j);
int fn2(int i);
```

```
/*-----------------------------------------------------------------*
  Global Variables
 *-----------------------------------------------------------------*/
int gint;

/*-----------------------------------------------------------------*
  Main Program
 *----------------------------------------------------------------*/
int main(void)
{
  int pid,x;
  int shared_local;
  printf( "Fork Testing!\r\n" );
  fflush( stdout );
  gint = 5;
  shared_local =5;

  pid = fork();
  if(pid == 0)
  {
    x=0;
    gint = 10;
    shared_local = fn1(9);
    printf("\nForked : Child");
    printf("\nChild:Global=%d,Shared_Local=%d",gint,shared_local);
    printf("\nChild pid = %d, parent pid =%d \n", getpid(),getppid());
    fflush( stdout );
  }
  else
  {
    x=0;
    gint = 12;
    shared_local = fn2(11);
    printf("\nForked : Parent");
    printf("\nParent:Global=%d,Shared_Local=%d",gint,shared_local);
    printf("\nParent pid = %d, Parent's parent pid =%d \n",
getpid(),getppid());
    fflush( stdout );

  }
  return 0;
}

int fn1(int j)
{
  j++;
  return j;
}

int fn2(int i)
{
  i++;
  return i;
}
```

19. Enter the source code shown in Listing 9.4 into the editor window of `Child.c` file.

Listing 9.4 Source Code for Exec-1.c

```
/**  Exec-1.c *
Demonstrates Exec system call functionality */
/*------------------------------------------------------------------*
System Include files
*------------------------------------------------------------------*/
#include <stdio.h>
#include <unistd.h>
/*------------------------------------------------------------------*
Main Program
*------------------------------------------------------------------*/
int main(void){
  printf("After exec my process ID is %d\n",getpid());
  printf("My parent process's ID is %d\n",getppid());
  printf("exec ends\n");
  fflush( stdout );
  return 0;
}
```

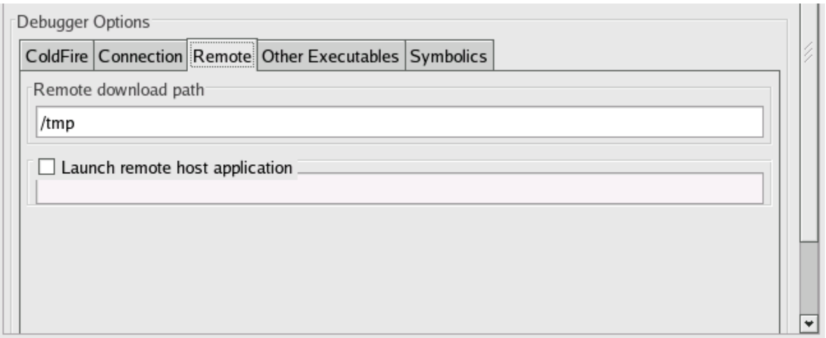
20. Build `Exec` project.
    a. Select the **Parent_debug** launch configuration, if not selected.
    b. Select **Project > Build Project**.

       The CodeWarrior IDE generates the `exec.elf`, and `exec-1.elf` executable files and places them in the project folder.
21. Specify the remote download path of the executable files to be launched by the `exec()` system call.
    a. Select **Run** > **Open Debug Dialog** to open the **Debug** window.
    b. In the left pane from the CodeWarrior Download container, select the **Parent_debug** launch configuration.
    c. On the Debugger page, click the Remote tab.

d. Specify `/tmp` in the **Remote Download Path** field as seen in Figure 9.12. This specifies that the final executable file will be downloaded to this location on the target platform for debugging.

NOTE For this example, the remote download path is specified as `/tmp`. If you want, you may specify an alternate remote download path for the executable file.

Figure 9.12 Remote Download Path — Shared Library Project



Remote download path: /tmp

☐ Launch remote host application

22. Specify the host-side location of the executable files to be launched by the `exec()` system call.
    a. Click the **Other Executables** tab in the **Debugger** page.
    b. Click **Add**.

       The **Debug Other Executable** dialog box appears.
    c. Click the **Workspace** button to locate the additional executable file.
    d. Navigate to the location of the `exec-1.elf` file in your project directory.
    e. Select the `exec-1.elf` filename.
    f. In **Relative To** list box, select **Project**.
    g. Click **Open**. The host-side location of `exec-1.elf` appears in the **Specify the location of the other executable** field.
    h. Check the **Load Symbols** checkbox.
    i. Check the **Download to Device** checkbox.

       The **Specify the remote download path** field is activated.

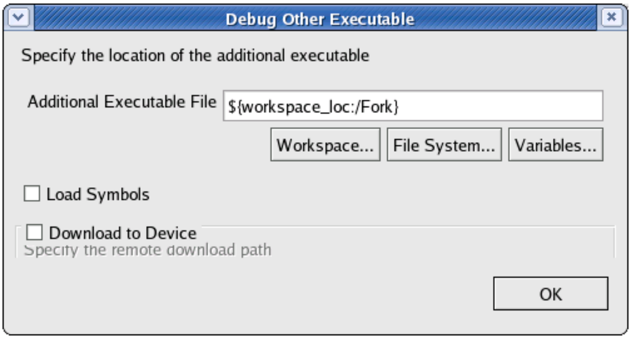
NOTE If you do not want to download the selected file on the target platform, do not check the **Download to Device** checkbox.

   j. Specify `/tmp` in the **Remote download path** text box. The shared library will be downloaded at this location when you debug or run the executable file.

   k. Click **OK**. The settings as seen in Figure 9.13 are saved.

   l. Click **Apply** to save the settings made to the launch configuration.

Figure 9.13 Debug Other Executable Dialog Box



Specify the location of the additional executable

Additional Executable File: ${workspace_loc:/Fork}

☐ Load Symbols

☐ Download to Device

Specify the remote download path

23. Set breakpoints in the child and parent processes.

    a. Double-click the `fork.c` filename in the **CodeWarrior Projects** view.

    b. Set a breakpoint in the code of the child process at this line:

       `x=0;`

    c. Set a breakpoint in the code of the parent process.

    d. Close the `fork.c` file.

24. Select **Project > Debug**.

    The debugger window appears.The debugger downloads the `exec.elf` and `exec-1.elf` executable files to the specified location on the remote target one by one.

25. Step over the code until you reach the line of code that calls the `fork()` system call:

    ```
    pid = fork ();
    ```

    When the `fork()` system call is called, the child process debugger window appears. You can now perform normal debugging operations in this window.

26. Step over the code in the child process debugger window a couple of times.

    When the `exec()` function call in the child process code executes, a new debugger window appears. This window displays the code of the `exec-1.elf` executable file. You can now perform normal debugging operations in this window.

27. Next, step over the code in the parent process debugger window a couple of times.

NOTE The console window of the parent process is shared by the child process.

Debugging Shared Library

This product lets you do source-level debugging of shared libraries. When you debug an executable file with which a shared library interacts, you can step into the shared library code.

This example demonstrates how to debug a shared library that is implicitly-linked to an application. The steps are:

1. Create an example shared library project. Refer Create Shared Library Project.
2. Build the example shared library project. Refer Build Shared Library Project.
3. Create and build an example application that links to the example shared library. Refer Configure Executable Build Target.
4. Configure the library launch configuration. Refer, Configure Library Build Target.
5. Debug the example shared library project. Refer, Debug Shared Library.

Create Shared Library Project

1. In the CodeWarrior IDE, use the **New Project** wizard to create a new project with the settings shown in Table 9.4.

Table 9.4 Shared Library Project Settings

Project Name	`SharedLib`
Project Location	`~/SharedLib`
Stationery Type	Linux Application
Languages	C

2. Remove the default `main.c` file and add the source files (`SharedLibImplicit.c` and `Library_Examples.c`) to your project.
3. In the CodeWarrior IDE create two header files, and save them in the project directory:
   - `LibExample.h`
   - `CWExample.h`

4. Enter the source code shown in Listing 9.5 into the editor window of the `LibExample.h` file.

Listing 9.5 Source Code for LibExample.h

```
/* LibExample.h */
int add_example(int x,int y);
int add_example_local(int x,int y)
```

5. Enter the source code shown in Listing 9.6 into the editor window of the `CWExample.h` file.

Listing 9.6 Source Code for CWExample.h

```
/* CWExample.h */
 #define INFINITE_LOOP     while(1);
```

6. Enter the source code shown in Listing 9.7 into the editor window of the `SharedLibImplicit.c` file.

Listing 9.7 Source Code for SharedLibImplicit.c

```
/*
  SharedlibImplicit.c
  Demonstrates implicit linking.
*/

// User Include files

#include “LibExample.h”
#include “CWExample.h”

// function prototype declaration

int temp (int, int);

// main program

int main ()
{
  int ret;
  int a, b;
  a = 10;
  b = 20;
  ret = temp (a, b);
  ret = add_example (a, b);   // step in here
  return ret;
}

int temp(int i, int j)
{
  return i + j;
}
```

7. Enter the source code shown in Listing 9.8 into the editor window of the `Library_Examples.c` file.

Listing 9.8 Source Code for Library_Examples.c

```
/*
  LibExample.c
*/

// user include files

#include "LibExample.h"

// functions definitions

int add_example( int x, int y)
{
  int p,q;
  p=100;
  q=p+200;
  add_example_local (2, 3);  // step in here
  return x+y+q;
}
  int add_example_local (int x, int y)
{
  int p,q;
  p = 100;
  q = p + 200;
  return x + y + q;
}
```

Now, let us generate the shared library application and debug it. The following sections describe how to debug a shared library:

- Build Shared Library Project
- Configure Executable Build Target
- Configure Library Build Target
- Debug Shared Library

Build Shared Library Project

You first need to build the project to generate the shared library file and the executable binary.

1. Build the `SharedLib` project
    a. Right-click the project name in the **CodeWarrior Projects** view to open the context menu.
    b. From the context menu, select the **LibExample** launch configuration by selecting **Build Configurations** > **Set Active** > <*Launch Configuration Name*>.

TIP You can also select a launch configuration from the drop-down list that appears when you click the down arrow next to the project name in the **CodeWarrior Projects** view.

    c. Select **Project > Build Project**.

    The CodeWarrior IDE builds the project and stores the output file `LibExample.so` in the *Output* directory within the project directory.

    d. Now, select the **SharedLib_IM** launch configuration.
    e. Select **Project > Build Project**.

    The CodeWarrior IDE builds the project and stores the final output file `SharedLib_IM.elf` in the *Output* directory within the project directory.

Configure Executable Build Target

You need to set up the **Application_debug** build target by:

- specifying the remote download path of the final executable file
- specifying the host-side location and the remote download path of the shared library
- specifying the environment variable that enables the shared object loader to locate the shared library on the remote target at runtime

1. Make the **Application_debug** build target in the **CodeWarrior Projects** view active, if it not already active.
2. Specify the linker settings.

a. Right-click the **Application_debug** launch configuration in the **CodeWarrior Projects** view.

b. Select **Properties** from the context menu.

The **Properties** window for the shared library project appears.

c. In the **Tool settings** page, select **ColdFire Linux Linker container > Libraries**.

d. In the **Libraries (-l)** panel, click **Add** ( ).

The **Enter Value** dialog box appears.

e. Enter the library filename in the **Libraries** field.

f. In the **Libraries (-l)** panel, click **Add** ( ).

The **Add directory path** dialog box appears.

g. Enter the library path in the **Directory** field.

NOTE These settings enable the CodeWarrior IDE linker to locate the shared library `libexample.so`. For detailed information on other linker command line arguments, refer GNU linker manuals.

3. Specify the remote download path of the final executable file.

a. Select **Run** > **Debug Configurations** to open the **Debug Configurations** window.

b. In the left pane, select **CodeWarrior Download** > **Application_debug** launch configuration.

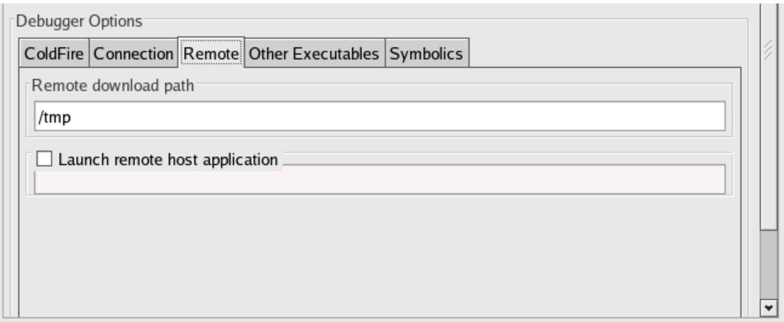
c. Click the **Debugger** tab.

d. Click the **Remote** tab.

e. Specify `/tmp` in the **Remote Download Path** field as seen in Figure 9.14. This specifies that the final executable file will be downloaded to this location on the target platform for debugging.

NOTE For this example, the remote download path is specified as `/tmp`. If you wish, you may specify an alternate remote download path for the executable file.

Figure 9.14 Remote Download Path — Shared Library Project



Remote download path
/tmp

☐ Launch remote host application

4. Specify the host-side location and the remote download path of the shared library.
   a. Click the **Other Executables** tab in the **Debugger** page.
   b. Click **Add**.

      The **Debug Other Executable** dialog box appears.
   c. Click **Workspace**.

      The **Select a file** dialog box appears.
   d. Navigate to the location where you have stored the `LibExample.so` file in your project directory.
   e. Select the `LibExample.so` filename.
   f. In **Relative To** list box, select **Project**.
   g. Click **Open**. The host-side location of the shared library appears in the **Specify the location of the other executable** field.
   h. Check the **Load Symbols** checkbox.
   i. Check the **Download to Device** checkbox.

      The **Specify the remote download path** field is activated.

NOTE If you do not want to download the selected file on the target platform, do not check the **Download to Device** checkbox.

   j. Specify `/tmp` in the **Remote download path** text box. The shared library will be downloaded at this location when you debug or run the executable file.

      The default location of shared libraries on the embedded Linux operating system is `/usr/lib`. For this example, the remote download location of `LibExample.so` is `/tmp`.
   k. Click **OK**.

5. Specify the environment variable that enables the shared object loader to locate the shared library on the remote target at runtime.

   At runtime, the shared object loader first searches for a shared library in the path specified by the `LD_LIBRARY_PATH` environment variable's value. In this case, the value of this environment variable will be `/tmp`, which is the remote download path for the shared library you specified in the **Debug Other Executable** dialog box. If you have not specified the environment variable or have assigned an incorrect value, the shared object loader searches for the shared library in the default location `/usr/lib`.

   a. In the **Debug** window, click **Environment** to open the **Environment** page.

   b. Click **New** to open the **New Environment Variable** dialog box.

   c. In the **Name** field, type `LD_LIBRARY_PATH`.

   d. In the **Value** field, type `/tmp`.

NOTE Make sure you type the same remote download path in the **Value** field that you specified in the **Debug Other Executable** dialog box.

   e. Click **OK**.

   The environment variable is added to the launch configuration.

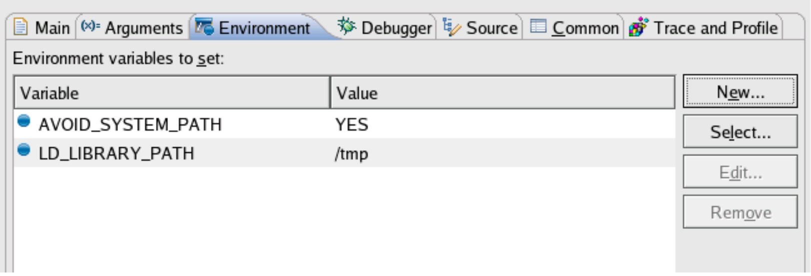
   f. Add another environment variable with name, `AVOID_SYSTEM_PATH` and value `YES`.

NOTE The `AVOID_SYSTEM_PATH` variable sets the launch configuration to use the library path settings you specify. By specifying the value `YES` you avoid the launch configuration from picking up any other system path.

   g. Click **Apply** to save the launch configuration settings.

   The target settings are saved as seen in Figure 9.15.

Figure 9.15 Environment Variables — Shared Library Project



Environment variables to set:

Variable	Value
AVOID_SYSTEM_PATH	YES
LD_LIBRARY_PATH	/tmp

   h. Click **OK** to close the **Debug Configurations** window.

6. Select **Project > Build Project** to build the project.

   The final executable is built with new launch configuration settings.

Configure Library Build Target

You need to configure the **LibExample** build target by:

- specifying the host-side location of the executable file to be used for debugging the shared library
- specifying remote debugging options

1. Make the **LibExample** launch configuration in the project active.
2. Specify the host-side location of the executable file to be used for debugging the shared library.
   a. Click the **Other Executables** tab in the **Debugger** page.
   b. Click **Add**.

      The **Debug Other Executable** dialog box appears.
   c. Click **Workspace**.

      The **Select a file** dialog box appears.
   d. Navigate to the location where you have stored the `LibExample.so` file in your project directory.
   e. Select the `LibExample.so` filename.
   f. In **Relative To** list box, select **Project**.
   g. Click **Open**.

      The host-side location of the shared library appears in the **Specify the location of the other executable** field.
   h. Check the **Load Symbols** checkbox.
   i. Check the **Download to Device** checkbox.

      The **Specify the remote download path** field is activated.
   j. Specify `/tmp` in the **Remote download path** text box.

      The shared library will be downloaded at this location when you debug or run the executable file.

      The default location of shared libraries on the embedded Linux operating system is `/usr/lib`. For this tutorial, the remote download location of `LibExample.so` is `/tmp`.
   k. Click **OK**.

3. Specify remote debugging options.
    a. On the **Debugger** page, click the **Remote** tab.
    b. Specify `/tmp` in the **Remote download path** field. This is the location where the shared library will be downloaded on the target for debugging.
    c. Check the **Launch remote host application** checkbox.
    d. Specify `SharedLib_IM/SharedLib_IM.elf` in the text box below the **Launch remote host application** checkbox.
    e. Click **Apply** to save the launch configuration settings.
    f. Close the **Debug Configurations** dialog box.
4. Build the project.

    Select **Project > Build Project**.

    The library is built with the new settings.

Debug Shared Library

In the steps that follow, you will launch the debugger. Next, you will step through the code of the executable file `SharedLib_Application.elf` until you reach the code that makes a call to the `add_example` function implemented in the shared library. At this point, you will step into the code of the `add_example` function to debug it.

1. Make the **Application_debug** build target in the **CodeWarrior Projects** view active.
2. Select **Run > Debug Configurations**.

    The debugger starts and downloads the `SharedLib_Application.elf` and `LibExample.so` files to the specified location on the remote target, one after another. The **Debug** perspective appears.
3. Step Over the code. Click the **Step Over** button in the debugger window until you reach this line of code: `ret=add_example(a,b);`.
4. Step into the code of the `add_example` function. In the **Debug** view, click the **Step Into** button a couple of times to step into the code of the `add_example` function.

    The debugger steps into the source code of the `add_example` function in the `Library_Examples.c` file.

5. Step through rest of the code. After stepping in, you can step through the rest of the code.
6. Run the rest of the application. Click the **Run** button.

    The rest of the code is executed and the output appears in the **CodeWarrior TRK Console** window.

Debugging U-Boot Loader

This section shows you how to use the CodeWarrior debugger to debug the U-Boot universal boot loader. U-Boot resides in flash memory on target systems and boots an embedded Linux image developed for those systems.

The topics in this section are:

- Installing Board Support Package
- Configuring Linux Target Image Builder
- Debugging U-Boot in Flash
- Debugging U-Boot in RAM
- Debugging Linux Kernel

Installing Board Support Package

1. On the Linux computer download the Board Support Package (BSP) for you target hardware to install Linux Target Image Builder (LTIB) kernel files and Linux compiler toolchains on your system.

   Board Support Package CD image files for test boards are located at `www.codewarrior.com` under:

   Downloads > Linux Board Support Packages > BSPs for ColdFire Architectures.

2. Download the BSP CD image file for your test board.

NOTE You will need to log-in or register to download the BSP CD image file.

   The downloaded CD image file has an `.iso` extension.

   For example, `M54455_20071001-LTIB.iso`

3. Mount the image file to the CDROM as root, or via sudo:

```
<sudo> mount -o loop M54455_20071001-LTIB.iso
/mnt/cdrom
```

NOTE `sudo` is a Linux utility that allows users privileges to run applications as `root`. You need to be setup to run `sudo` commands by your system administrator.

4. Execute the BSP install file to install the LTIB files to a directory of your choice, where you have privileges to write files:

```
/mnt/cdrom/install
```

NOTE The BSP install must be installed as a non-root user, otherwise the install will exit.

5. Answer the questions from the installation program until the file copy process begins.

   The Linux Target Image Builder (LTIB) files are copied to the installation directory.

   For example, `/home/username/M54455_20071001-LTIB`

Configuring Linux Target Image Builder

The next step in the process is to configure Linux Target Image Builder (LTIB) to build uboot with debug symbols.

Installation includes copying files to the `/opt` directory. This directory is owned by root so you need `sudo` privileges to run `rpm install`.

1. Add, or have your system administrator add, the following command to the `sudoers` text file:

```
<username> ALL = NOPASSWD: /bin/rpm, /opt/freescale/ltib/
usr/bin/rpm
```

NOTE The `sudoers` text file is edited using `usr/sbin/visudo`

2. Change the directory to the LTIB subdirectory.

   For example, `cd /home/<username>/ltib-M54455-20071001`

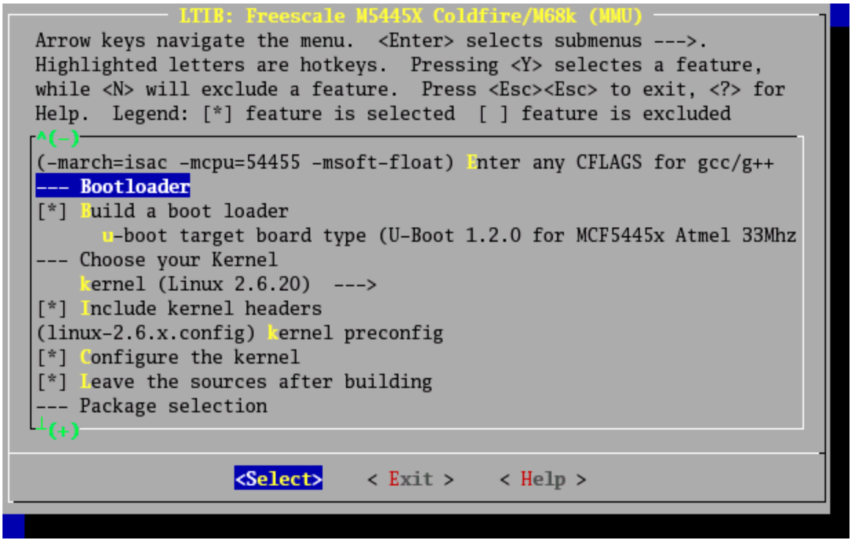
3. Run the command:

```
./ltib --configure
```

   This command will launch the LTIB shell script which will begin the installation of the LTIB files and launch the LTIB configuration screen. (Refer Figure 9.17.)

NOTE A log file named `host_config.log` contains a record of the install progress. This file can be viewed with the `tail -f host_config.log` command.

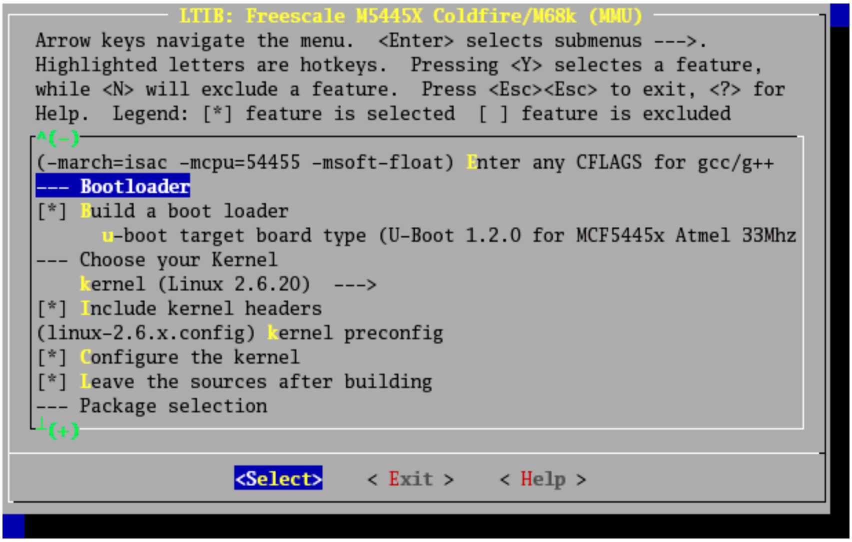
Figure 9.16 LTIB Configuration Screen



```
LTIB: Freescale M5445X Coldfire/M68k (MMU)
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help.  Legend: [*] feature is selected  [ ] feature is excluded
^(-)
(-march=isac -mcpu=54455 -msoft-float) Enter any CFLAGS for gcc/g++
--- Bootloader
[*] Build a boot loader
      u-boot target board type (U-Boot 1.2.0 for MCF5445x Atmel 33Mhz
--- Choose your Kernel
    kernel (Linux 2.6.20)  --->
[*] Include kernel headers
(linux-2.6.x.config) kernel preconfig
[*] Configure the kernel
[*] Leave the sources after building
--- Package selection
 (+)

              <Select>    < Exit >    < Help >
```

4. Configuration-screen head **Choose your toolchain** introduces the possible toolchains, select one that is appropriate.
5. Configuration-screen head **Board selection,** if available, introduces the possible processor boards - select one that is appropriate.
6. Select the **Build a boot loader** option.
7. Select the **U-Boot options** option. Refer Figure 9.17.

Figure 9.17 LTIB Configuration Screen



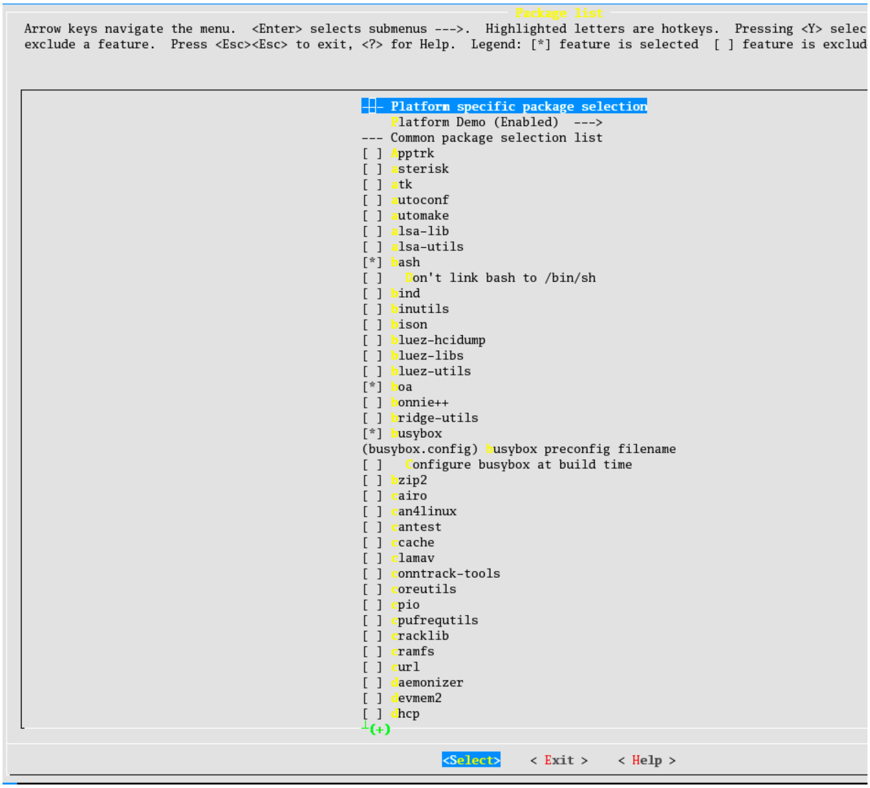
```
LTIB: Freescale M5445X Coldfire/M68k (MMU)
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help.  Legend: [*] feature is selected  [ ] feature is excluded
^(-)
(-march=isac -mcpu=54455 -msoft-float) Enter any CFLAGS for gcc/g++
--- Bootloader
[*] Build a boot loader
      u-boot target board type (U-Boot 1.2.0 for MCF5445x Atmel 33Mhz
--- Choose your Kernel
    kernel (Linux 2.6.20)  --->
[*] Include kernel headers
(linux-2.6.x.config) kernel preconfig
[*] Configure the kernel
[*] Leave the sources after building
--- Package selection
↓(+)

            <Select>    < Exit >    < Help >
```

8. Select the **Configure the kernel** option.
9. Select the **Leave the sources after building** option.
10. Select the **Package** list option.

    The package list screen appears.

11. Select the required packages. For example, select apptrk if you plan to do application debug or you may deselect packages not required if you plan to have a smaller ramdisk.

Figure 9.18 LTIB Package List Screen



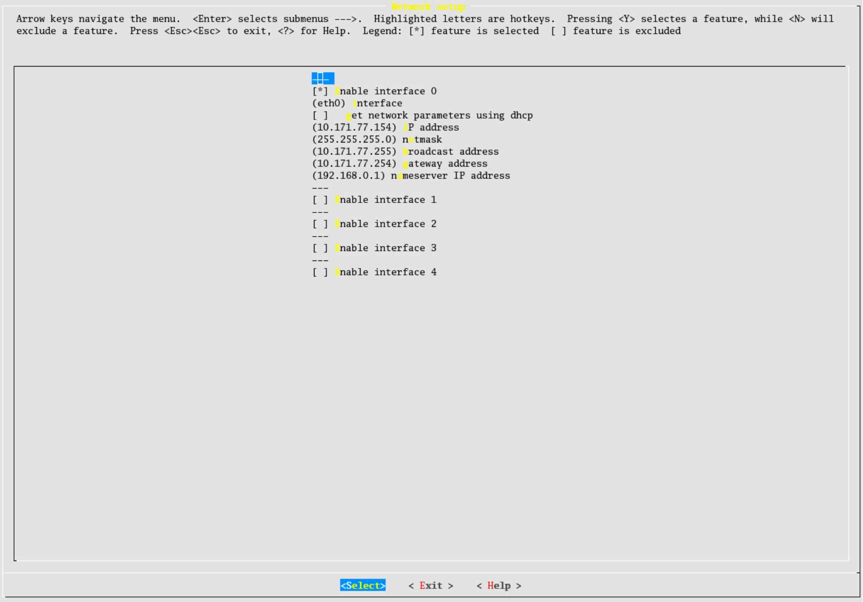
```
                                      Package list
 Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> selec
 exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] feature is selected  [ ] feature is exclud

                                              --- Platform specific package selection
                                                  Platform Demo (Enabled)  --->
                                              --- Common package selection list
                                              [ ] Apptrk
                                              [ ] asterisk
                                              [ ] atk
                                              [ ] autoconf
                                              [ ] automake
                                              [ ] alsa-lib
                                              [ ] alsa-utils
                                              [*] bash
                                              [ ]   Don't link bash to /bin/sh
                                              [ ] bind
                                              [ ] binutils
                                              [ ] bison
                                              [ ] bluez-hcidump
                                              [ ] bluez-libs
                                              [ ] bluez-utils
                                              [*] boa
                                              [ ] bonnie++
                                              [ ] bridge-utils
                                              [*] busybox
                                              (busybox.config) busybox preconfig filename
                                              [ ]   Configure busybox at build time
                                              [ ] bzip2
                                              [ ] cairo
                                              [ ] can4linux
                                              [ ] cantest
                                              [ ] ccache
                                              [ ] clamav
                                              [ ] conntrack-tools
                                              [ ] coreutils
                                              [ ] cpio
                                              [ ] cpufrequtils
                                              [ ] cracklib
                                              [ ] cramfs
                                              [ ] curl
                                              [ ] daemonizer
                                              [ ] devmem2
                                              [ ] dhcp
                                              (+)

                                                   <Select>    < Exit >    < Help >
```

12. Select the **Platform specific package selection** option.

13. Select the required options and modify them.

14. Select the **Network setup** option if you want to configure the network. Refer Figure Figure 9.19.

Figure 9.19 LTIB Network Setup Options

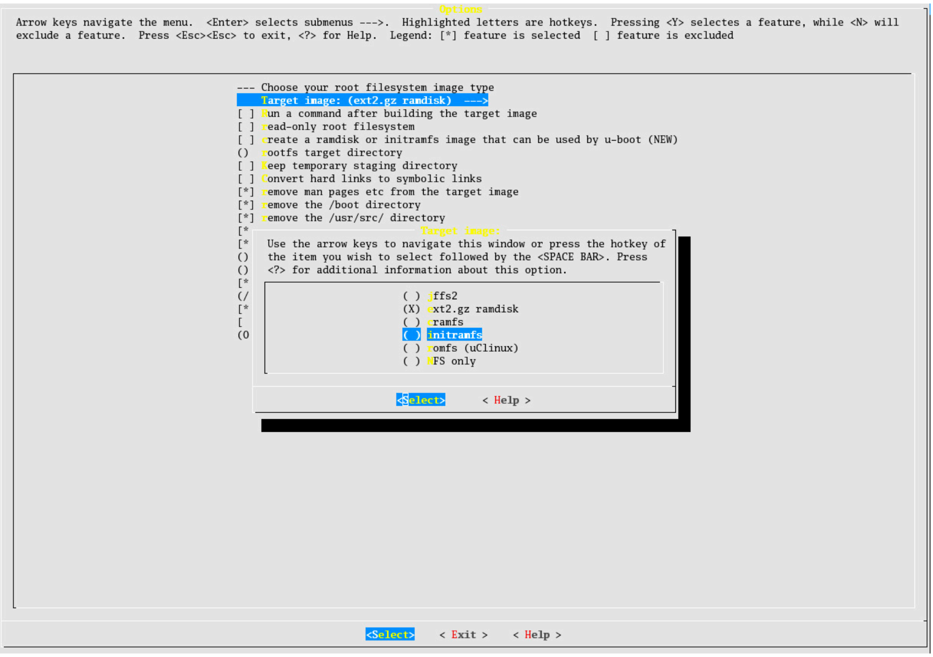


```
                                   Network setup
 Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while <N> will
 exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] feature is selected  [ ] feature is excluded

                                                 ---
                                                 [*] Enable interface 0
                                                 (eth0) interface
                                                 [ ]   get network parameters using dhcp
                                                 (10.171.77.154) IP address
                                                 (255.255.255.0) netmask
                                                 (10.171.77.255) broadcast address
                                                 (10.171.77.254) gateway address
                                                 (192.168.0.1) nameserver IP address
                                                 ---
                                                 [ ] Enable interface 1
                                                 ---
                                                 [ ] Enable interface 2
                                                 ---
                                                 [ ] Enable interface 3
                                                 ---
                                                 [ ] Enable interface 4

                                      <Select>    < Exit >    < Help >
```

15. Select **Target image**.

16. Ensure that **Create a ramdisk that can be used by u-boot** is selected. Refer Figure 9.20.

Figure 9.20 LTIB RAM Disk Creation Option



```
                                   Options
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> selectes a feature, while <N> will
exclude a feature.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] feature is selected  [ ] feature is excluded

                                   --- Choose your root filesystem image type
                                       Target image: (ext2.gz ramdisk)  --->
                                   [ ] Run a command after building the target image
                                   [ ] read-only root filesystem
                                   [ ] create a ramdisk or initramfs image that can be used by u-boot (NEW)
                                   ()  rootfs target directory
                                   [ ] Keep temporary staging directory
                                   [ ] Convert hard links to symbolic links
                                   [*] remove man pages etc from the target image
                                   [*] remove the /boot directory
                                   [*] remove the /usr/src/ directory
                                   [*                   Target image:
                                   [*   Use the arrow keys to navigate this window or press the hotkey of
                                   ()   the item you wish to select followed by the <SPACE BAR>. Press
                                   ()   <?> for additional information about this option.
                                   [*
                                   (/                       ( ) jffs2
                                   [*                       (X) ext2.gz ramdisk
                                   [                        ( ) cramfs
                                   (0                       ( ) initramfs
                                                            ( ) romfs (uClinux)
                                                            ( ) NFS only

                                                       <Select>      < Help >

                                         <Select>    < Exit >    < Help >
```

17. Select **Exit.**

18. Select **Yes** when you are asked if you want to save your new configuration.

    The installation process continues.

NOTE The LTIB installation process creates a sub-directory in the `/opt` directory - `/opt/freescale`. The Linux toolchains and kernel code are located in this

---

a. Make these parameter additions to these variables in file *LtibInstallDir*`/rpm/BUILD/`*`u-boot`*`/config.mk`:

   - `DBGFLAGS = -g2 -gdwarf-2`
   - `AFLAGS_DEBUG = -Wall -gdwarf-2`
   - `OPTFLAGS = -01`

NOTE If you are using a Linux Target Image Builder (LTIB) BSP, you may need to change the optimization flag of the `CFLAGS` argument in this file: *`LTIBInstallDir`*`/config/platform/`*`board_name`*`/.config`

---

19. Edit the following file:

    LtibInstallDir`/rpm/BUILD/u-boot/lib_m68k/board.c`

    and change the line `debug` to `printf` on the line "`Now running in RAM ...`"

20. Rebuild U-Boot using the following command:

    ```
    ./ltib -m scbuild -p u-boot-1.2.0-mcf5445x.spec
    ```

    U-Boot builds to completion.

21. Deploy U-Boot using the following command:

    ```
    ./ltib -m scdeploy -p u-boot-1.2.0-mcf5445x.spec
    ```

    This command copies U-Boot to appropriate locations.

    In the *LTIBInstallDir*`/rpm/BUILD/u-boot` directory you now have a U-Boot binary executable file that contains debugger symbolic information (`u-boot.srec`). In addition, you have a U-Boot raw binary file (`u-boot.bin`) and a U-Boot ELF-format binary executable file (`u-boot`).

TIP It is best to leave the ELF-format U-Boot binary file in this location, so that when you create a CodeWarrior project with the file, the CodeWarrior IDE can find all of the source U-Boot source code files needed to debug the binary file.

Creating CodeWarrior Project to Debug U-Boot

1. Start the CodeWarrior IDE.
2. Select **File > Import**.

   The **Import** wizard appears.
3. Expand the **C/C++** group and select **MCU Executable**.
4. Click **Next**.
5. Specify a name in the **New Project Name** text box.
6. Click **Next**.
7. In the **File to import** text box, specify the path `D:\temp\u-boot-1.3.2\u-boot`.
8. Click **Next**.
9. Select the device.
10. Click **Next**.
11. Select the connection.
12. Click **Finish**.

13. Right-click on the project and select **Debug As > Debug Configurations**.

    The **Debug Configurations** dialog box appears.

14. Specify the launch configuration settings as in Table 9.5.

Table 9.5 U-Boot Project Launch Configuration Settings

Debug Window Component	Settings
Debugger Page > ColdFire	Target Processor > *<processor>*
	Execute Reset — Disabled
	Target Initialization File — None
	Memory Configuration File — Disabled
Debugger Page > Connection Tab	Connection Protocol — CCS
	Physical connection — Generic
Debugger Page > PIC Tab	Alternate Load Address — 0xfff80000

15. Click the **Source** tab. Enter the source mapping configuration by following these steps.

    a. Click **Add**.

       The **Add Source** dialog box appears.

    b. Select **Path Mapping**.

    c. Click **OK**.

       A new path mapping container appears in the **Source Lookup Path** area.

    d. Select `Path Mapping: New Mapping` and click **Edit**.

       The **Path Mappings** dialog box appears.

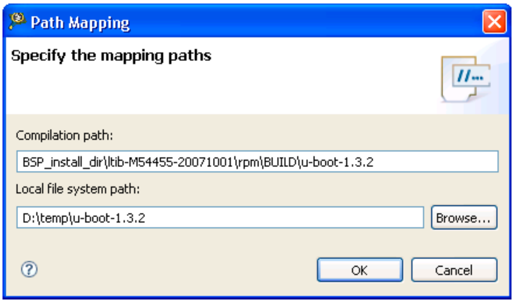
    e. Click **Add**.

       The **Path Mapping** dialog box appears.

    f. Enter the following values in the **Compilation path** and **Local file system path**.

       - Compilation Path: `BSP_install_dir\ltib-M54455-20071001\rpm\BUILD\u-boot-1.3.2`
       - Local file system path: `D:\temp\u-boot-1.3.2`

Figure 9.21 Path Mappings Dialog Box


Path Mapping
Specify the mapping paths

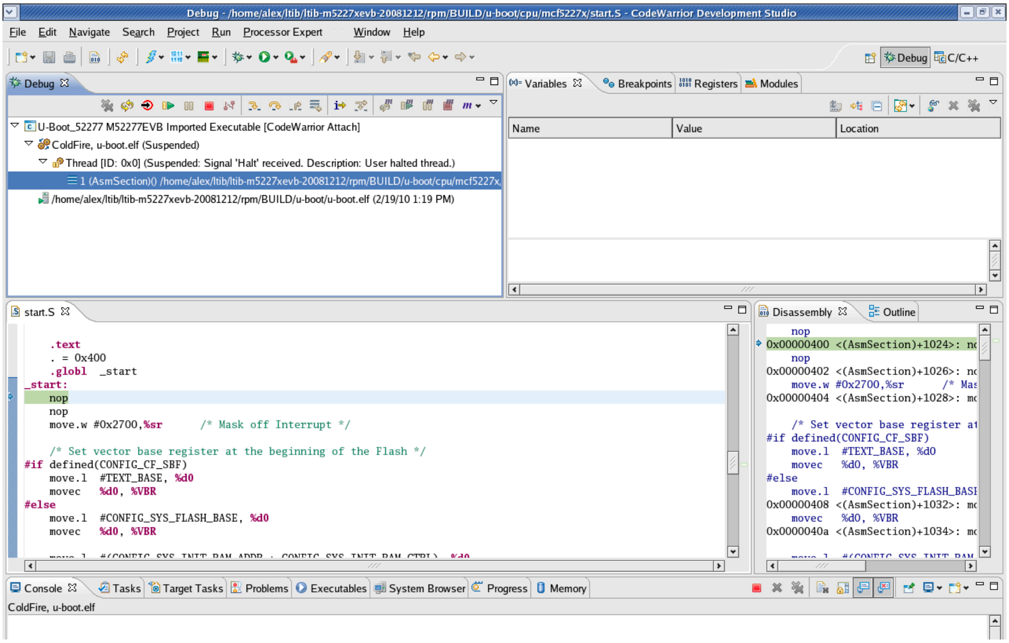
Compilation path: BSP_install_dir\ltib-M54455-20071001\rpm\BUILD\u-boot-1.3.2

Local file system path: D:\temp\u-boot-1.3.2

Browse...
OK
Cancel

16. Click **Apply** to save the settings made to the various tabs.
17. Click **Debug**.
18. This debugs the U-boot.

Figure 9.22 Debug Perspective — U-Boot Debug


Debug - /home/alex/ltib/ltib-m5227xevb-20081212/rpm/BUILD/u-boot/cpu/mcf5227x/start.S - CodeWarrior Development Studio
File Edit Navigate Search Project Run Processor Expert Window Help
Debug C/C++
Debug
Variables Breakpoints Registers Modules
Name Value Location
start.S
Disassembly Outline
Console Tasks Target Tasks Problems Executables System Browser Progress Memory

```
U-Boot_52277 M52277EVB Imported Executable [CodeWarrior Attach]
  ColdFire, u-boot.elf (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      1 (AsmSection)() /home/alex/ltib/ltib-m5227xevb-20081212/rpm/BUILD/u-boot/cpu/mcf5227x,
  /home/alex/ltib/ltib-m5227xevb-20081212/rpm/BUILD/u-boot/u-boot.elf (2/19/10 1:19 PM)
```

```
    .text
    . = 0x400
    .globl  _start
_start:
    nop
    nop
    move.w #0x2700,%sr      /* Mask off Interrupt */

    /* Set vector base register at the beginning of the Flash */
#if defined(CONFIG_CF_SBF)
    move.l  #TEXT_BASE, %d0
    movec   %d0, %VBR
#else
    move.l  #CONFIG_SYS_FLASH_BASE, %d0
    movec   %d0, %VBR
```

```
    nop
0x00000400 <(AsmSection)+1024>: nc
    nop
0x00000402 <(AsmSection)+1026>: nc
    move.w #0x2700,%sr      /* Mas
0x00000404 <(AsmSection)+1028>: mc

    /* Set vector base register at
#if defined(CONFIG_CF_SBF)
    move.l  #TEXT_BASE, %d0
    movec   %d0, %VBR
#else
    move.l  #CONFIG_SYS_FLASH_BASI
0x00000408 <(AsmSection)+1032>: mc
    movec   %d0, %VBR
0x0000040a <(AsmSection)+1034>: mc
```

ColdFire, u-boot.elf

During a typical U-Boot start-up sequence, the target processor starts executing U-Boot in Flash memory. U-Boot then enables the Memory Management Unit (MMU), and relocates itself to RAM.

CodeWarrior build target settings required to debug U-Boot in Flash memory differ from the settings required to debug U-Boot in RAM. Therefore, you must use individual CodeWarrior debug sessions to debug the Flash memory and RAM sections:

- Debugging U-Boot in Flash

  This procedure shows how to debug U-Boot in Flash memory.

- Debugging U-Boot in RAM

  This procedure shows how to debug U-Boot in RAM.

Debugging U-Boot in Flash

The first stage is when u-boot runs in flash.

1. Enable **Debugger PIC Settings-> Alternate Load Addresses: <flash base address>**.
2. Start u-boot on the board,
3. Create an attach configuration with settings in Table 9.5.
4. Start debug.
5. Stop u-boot.
6. In the **Debug** view, click **Reset** to reset the board.
7. You are now at the __start section.

Debugging U-Boot in RAM

1. Enable **Debugger PIC Settings-> Alternate Load Addresses: <flash base address>**.

NOTE The second stage starts when you reach `relocate_code()` in `lib_m68k/board.c`. For this stage, you switch to RAM and do not execute code in flash. In order to have source correspondence, to see not only assembly, **PIC settings-> Alternate Address** must be enabled with the value U-boot prints: "`Now running in RAM - U-Boot at <some address>`" from which you have to substract `0x400`.

2. Start u-boot on the board.
3. Start debug.
4. Stop u-boot.
5. In the **Debug** view, click **Reset** to reset the board.
6. Set a breakpoint at `board_init_r()`.
7. Select Run, the breakpoint is encountered.
8. Click **Debug**.

NOTE Debugging optimized code not supported by Debugger: U-boot is compiled with `-O1` flags. However, the blue cursor might appear strange as no source correspondence is available.

Debugging Linux Kernel

This section shows you how to use the CodeWarrior debugger to debug the Linux kernel.

The Linux operating system (OS) works in two modes—*kernel mode and user mode*. The Linux kernel operates in kernel mode and resides in the top level of the OS memory space, *or kernel space*. The kernel performs the function of a mediator among all the currently running programs and between the programs and the hardware. The kernel manages the memory for all the programs (processes) currently running and ensures that the processes share the available memory such that each process has enough memory to function adequately. In addition, the kernel allows application programs to manipulate various hardware architectures via a common software interface.

User mode uses the memory in the lowest level of the OS memory space, called the user space or the application level. At the application level, a program accesses memory or other hardware through system calls to the kernel as it does not have permission to directly access these resources.The kernel debugging setup consists of several general actions, in this order:

- Checking elements of the CodeWarrior IDE installation: Checking CodeWarrior IDE Installation
- Connecting the test board to the Linux host: Setting Up Target Hardware
- Installing the board support package (BSP): Installing Board Support Package
- Configuring LTIB: Configuring LTIB
- Configuring and Compile the Kernel: Configuring Kernel
- Creating a CodeWarrior project with the kernel: Creating CodeWarrior Project

Checking CodeWarrior IDE Installation

The CodeWarrior IDE must be installed on the Linux host machine, if it is not already done.

NOTE The CodeWarrior installation instructions are located in the Microcontrollers V10.x Quick Start.

There are two places in the Codewarrior installation that may affect kernel debugging:

- GCC toolchain locations, refer
- CodeWarrior Platform Edition licensing, refer

GCC Toolchains

The GCC toolchains are the Linux tools used to compile embedded Linux applications. ColdFire systems are of two types: MMU enabled and MMU-less. For the MMU enabled systems, the embedded Linux is a regular GNU Linux. For MMU-less systems it is used a lighter version called uCLinux. Therefore, there are two types of toolchains supported by CodeWarrior. The paths to these toolchains must be passed to CodeWarrior using the `cwide-env` file in the product folder. Here is a sample of the section defining the toolchains:

```
CFGCCInstallBinDir= -add %CD%/../Cross_Tools/CodeSourcery_Linux/bin
```

```
CFGCCUCInstallBinDir= -add %CD%/../Cross_Tools/CodeSourcery_uClinux/bin
```

You can change the `cwide-env` file to point to match the actual locations of the GCC toolchains.

Setting Up Target Hardware

Before you can use the CodeWarrior IDE to debug the kernel, you need to have a debug probe connected between the Linux host and the target board.

Connect the hardware debug probe between target board and the Linux host. There are a variety of debug probes. The kernel debugging tutorial uses the USB Tap. Determine connection information for other debug probes from documentation provided with the probes.

USB Tap Connections

1. Ensure that the power switch on the target board is OFF.
2. Connect the square end (USB “B” connector) of the USB cable to the USB Tap.
3. Connect the rectangular end (USB “A” connector) of the USB cable to a free USB port on the host Linux machine.
4. Connect the ribbon cable coming out of the USB Tap to the 26-pin connector on the target board.
5. Establish a console connection before applying power to the board, so that boot messages can be viewed in a terminal window. Refer Establishing Console Connection.

Establishing Console Connection

You need to establish a console connection to the target hardware in order to:

- view debugging messages.
- confirm successful installation of the bootloader (`u-boot`).
- use the bootloader to boot the Linux OS.
- halt the booting of the Linux OS.

The bootloader receives keyboard input through a serial port that has default settings 115,200-8-N-1.

Follow these steps to establish a console connection to the target hardware.

1. Connect a serial cable from a serial port of the Linux host computer to a serial port of the target board.
2. On the Linux host computer, open a terminal-emulator program of your choice (ex. `minicom`).
3. From the terminal-emulator program, open a console connection to the target hardware, using the connection settings shown in Table 9.6.

Table 9.6 Terminal Connection Settings

Name	Setting
Baud rate	115,200 bits per second
Data bits	8
Parity	None
Stop bits	1
Flow control	Hardware

NOTE Refer to the board-specific README file inside the stationery wizard project to find out more details on the serial connection settings, changing the serial port on the board and the type of serial cable to use.

4. Test the connection by turning on the test board with the power switch and viewing the boot messages in the console connection.

Installing Board Support Package

You must download the Board Support Package (BSP) for your target hardware to install Linux Target Image Builder (LTIB) kernel files and Linux compiler toolchains on your system.

Find the Board Support Package CD image files for test boards at **www.codewarrior.com** under:

Downloads > Linux Board Support Packages > BSPs for Coldfire Architectures.

1. Download the BSP CD image file for your test board.

NOTE You will need to log-in or register to download the BSP CD image file.

   The downloaded CD image file has an `.iso` extension.

   Example: `M54455_20071001-LTIB.iso`

2. Mount the image file to the CDROM as root, or by using “`sudo`”:

```
<sudo> mount -o loop M54455_20071001-LTIB.iso
/mnt/cdrom
```

NOTE `sudo` is a Linux utility that allows users privileges to run applications as root. Your system administrator needs to set you up to run `sudo` commands.

3. Execute the BSP install file to install the LTIB files to a directory of your choice, where you have privileges to write files:

```
/mnt/cdrom/install
```

NOTE The BSP install must be installed as a non-root user, otherwise the install will exit.

4. Answer the questions from the installation program until the file copy process begins.

   The installation program copies the LTIB files to the installation directory.

   Example: `/home/`*`Username`*`/ltib-mcf5445x-20071001`

Configuring LTIB

The next step in the process is to configure Linux Target Image Builder (LTIB) to build the kernel with debug symbols.

Installation includes copying files to the `/opt` directory. This directory is owned by root, so you need `sudo` privileges to run `rpm install`.

1. Add, or have your system administrator add, the following command to the `sudoers` text file:

   `<username> ALL = NOPASSWD: /bin/rpm, /opt/freescale/ltib/usr/bin/rpm`

NOTE The `sudoers` text file is edited using `usr/sbin/visudo`

2. Change the directory to the LTIB subdirectory.

   Example: `cd /home/<username>/ltib-mcf5445x-20071001`

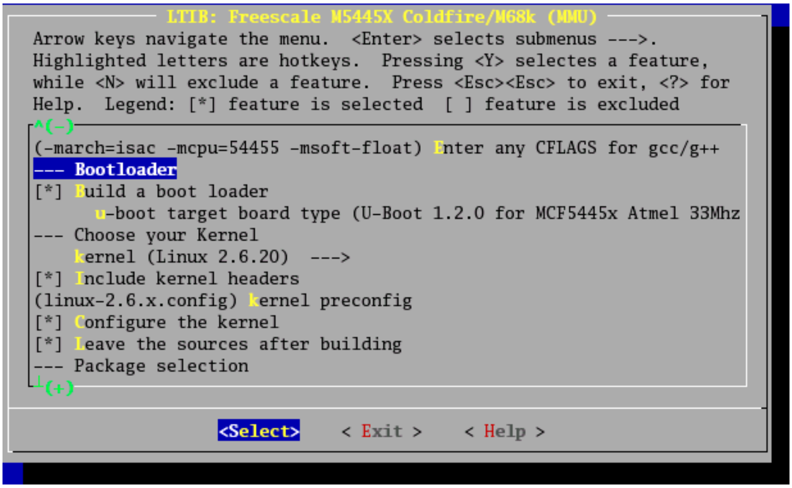
3. Run the command:

   `./ltib --configure`

   This command launches the LTIB shell script which begins the installation of the LTIB files and launches the LTIB configuration screen. Refer Figure 9.23.

NOTE A log file named `host_config.log` contains a record of the install progress. You can view this file with the `tail -f host_config.log` command.

Figure 9.23 LTIB Configuration Screen



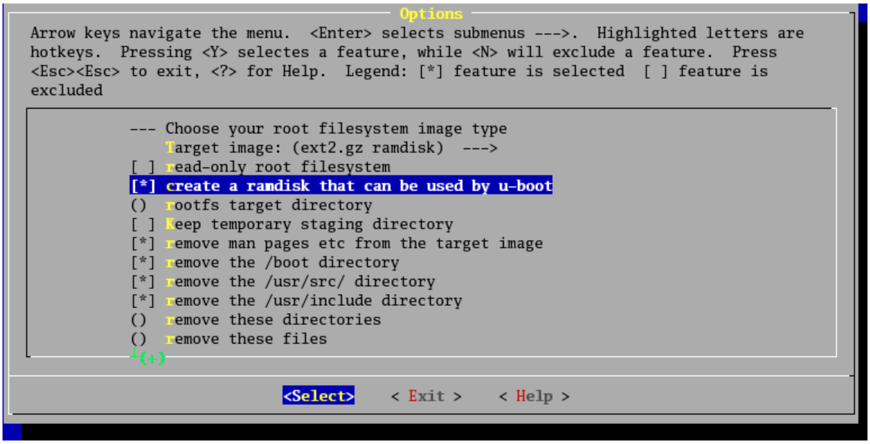
```
LTIB: Freescale M5445X Coldfire/M68k (MMU)
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help.  Legend: [*] feature is selected  [ ] feature is excluded
^(-)
(-march=isac -mcpu=54455 -msoft-float) Enter any CFLAGS for gcc/g++
--- Bootloader
[*] Build a boot loader
      u-boot target board type (U-Boot 1.2.0 for MCF5445x Atmel 33Mhz
--- Choose your Kernel
    kernel (Linux 2.6.20)  --->
[*] Include kernel headers
(linux-2.6.x.config) kernel preconfig
[*] Configure the kernel
[*] Leave the sources after building
--- Package selection
↓(+)

          <Select>    < Exit >    < Help >
```

4. Configuration-screen head **Choose your toolchain** introduces the possible toolchains - select one that is appropriate.

5. Make sure **Build a boot loader** is checked.

6. Configuration-screen head **u-boot target board type**, under **Build a boot loader,** introduces the possible processor boards. Select one that is appropriate.
7. Select the **Configure the kernel** option. Refer Figure 9.24.
8. Select the **Leave the sources after building** option. Refer Figure 9.24.
9. Scroll down to the **Target Image Generation > Options** heading.
10. Select the **Target Image Generation > Options** heading.
11. Select the type of kernel.
12. Select the filesystem image. Refer Figure 9.25.

Figure 9.24 LTIB RAM Disk Creation Option



```
Options
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are
hotkeys.  Pressing <Y> selectes a feature, while <N> will exclude a feature.  Press
<Esc><Esc> to exit, <?> for Help.  Legend: [*] feature is selected  [ ] feature is
excluded

           --- Choose your root filesystem image type
               Target image: (ext2.gz ramdisk)  --->
           [ ] read-only root filesystem
           [*] create a ramdisk that can be used by u-boot
           ()  rootfs target directory
           [ ] Keep temporary staging directory
           [*] remove man pages etc from the target image
           [*] remove the /boot directory
           [*] remove the /usr/src/ directory
           [*] remove the /usr/include directory
           ()  remove these directories
           ()  remove these files
           (+)

                 <Select>    < Exit >    < Help >
```

13. Select **Exit.**
14. Select **Yes** when you are asked if you want to save your new configuration.

    The installation process continues and brings up the kernel configuration screen. Refer Configuring Kernel.

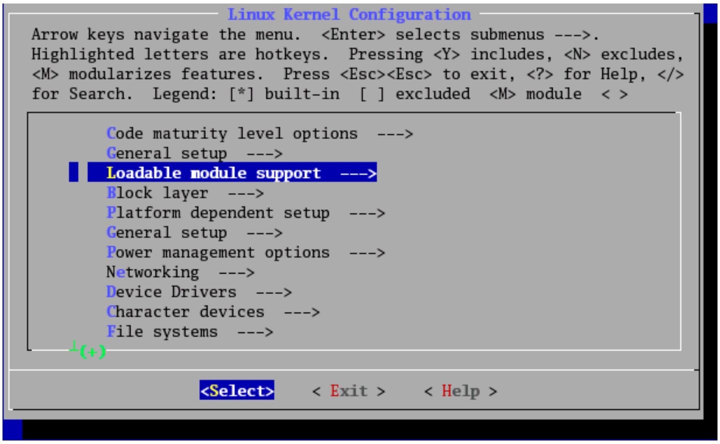
NOTE The LTIB installation process creates a sub-directory in the `/opt` directory - `/opt/freescale`. This directory contains the Linux toolchains and kernel code.

Configuring Kernel

After you complete the LTIB configuration, the kernel configuration screen appears. Refer Figure 9.26.

NOTE The kernel configuration screen displays only if **Configure the kernel** is selected in the LTIB configuration screen.

Figure 9.25 Loadable Module Support Menu

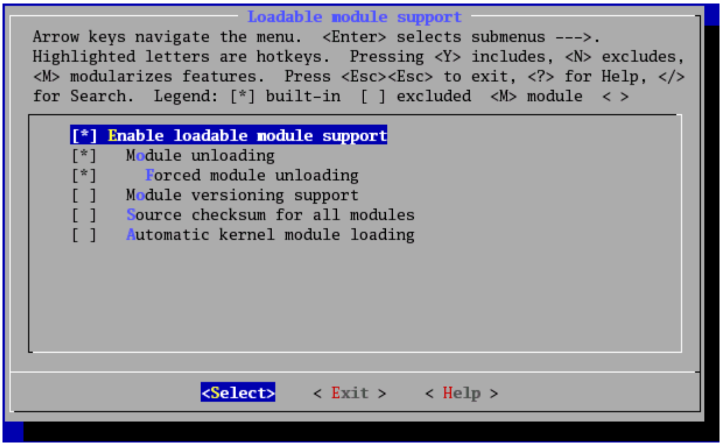


```
Linux Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

        Code maturity level options  --->
        General setup  --->
        Loadable module support  --->
        Block layer  --->
        Platform dependent setup  --->
        General setup  --->
        Power management options  --->
        Networking  --->
        Device Drivers  --->
        Character devices  --->
        File systems  --->
    (+)

           <Select>    < Exit >    < Help >
```

1. Select the **Loadable module support** menu item.

   The loadable module support options appear. Refer Figure 9.26.

Figure 9.26 Loadable Module Support Options

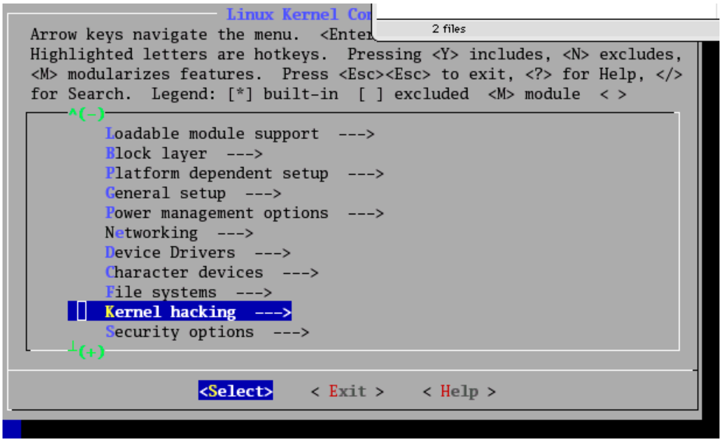


```
Loadable module support
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    [*] Enable loadable module support
    [*]   Module unloading
    [*]     Forced module unloading
    [ ]   Module versioning support
    [ ]   Source checksum for all modules
    [ ]   Automatic kernel module loading

           <Select>    < Exit >    < Help >
```

2. Select the **Enable loadable module support** option.

3. Select the **Module unloading** option.
4. Select **Exit** to return to the main configuration menu.
5. Select the **Kernel Hacking** menu item. Refer Figure 9.27.

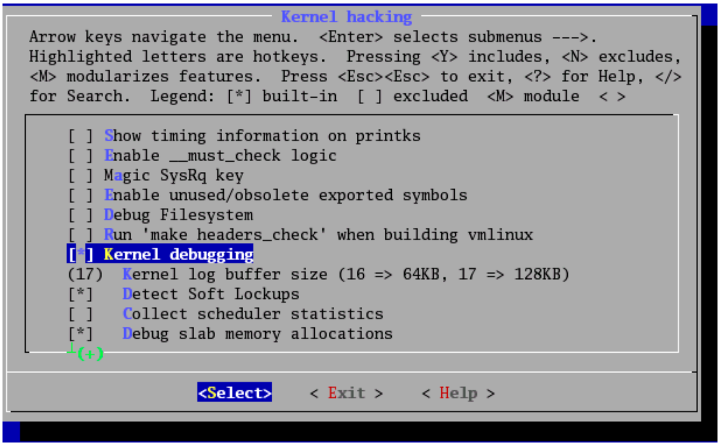
Figure 9.27 Kernel Hacking Menu



```
                          Linux Kernel Co
  Arrow keys navigate the menu.  <Enter
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
 ┌────^(-)──────────────────────────────────────────────────────────────┐
 │          Loadable module support  --->                               │
 │          Block layer  --->                                           │
 │          Platform dependent setup  --->                              │
 │          General setup  --->                                         │
 │          Power management options  --->                              │
 │          Networking  --->                                            │
 │          Device Drivers  --->                                        │
 │          Character devices  --->                                     │
 │          File systems  --->                                          │
 │      [ ] Kernel hacking  --->                                        │
 │          Security options  --->                                      │
 └────↓(+)──────────────────────────────────────────────────────────────┘
               <Select>    < Exit >    < Help >
```

The kernel debugging options appears. Refer Figure 9.28.

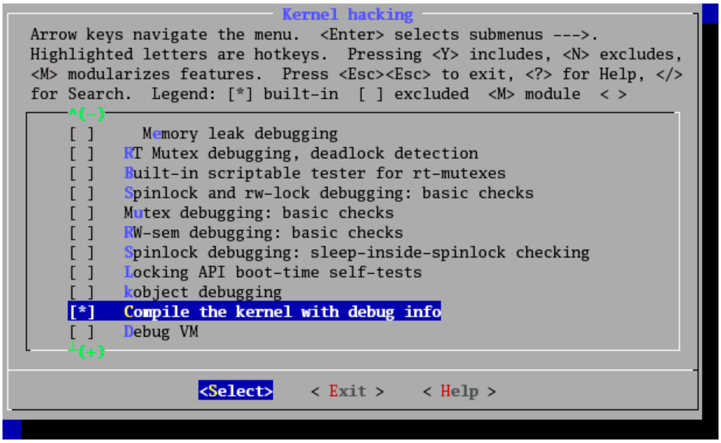
Figure 9.28 Kernel Hacking Options



```
                               Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
 ┌──────────────────────────────────────────────────────────────────────┐
 │      [ ] Show timing information on printks                          │
 │      [ ] Enable __must_check logic                                   │
 │      [ ] Magic SysRq key                                             │
 │      [ ] Enable unused/obsolete exported symbols                     │
 │      [ ] Debug Filesystem                                            │
 │      [ ] Run 'make headers_check' when building vmlinux              │
 │      [*] Kernel debugging                                            │
 │      (17)  Kernel log buffer size (16 => 64KB, 17 => 128KB)          │
 │      [*]   Detect Soft Lockups                                       │
 │      [ ]   Collect scheduler statistics                              │
 │      [*]   Debug slab memory allocations                             │
 └────↓(+)──────────────────────────────────────────────────────────────┘
               <Select>    < Exit >    < Help >
```

6. Select the **Kernel debugging** option. Refer Figure 9.28.
7. Select the **Compile the kernel with debug info** option. Refer Figure 9.29

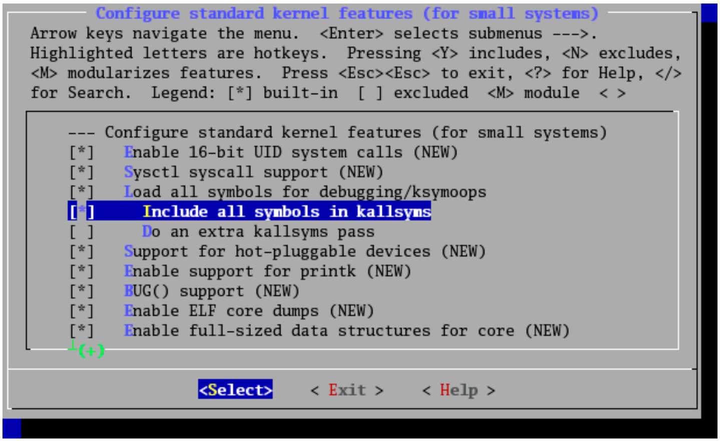
Figure 9.29 Kernel Hacking - Compile Kernel with Debug Info



```
                         Kernel hacking
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
    ^(-)
    [ ]     Memory leak debugging
    [ ]   RT Mutex debugging, deadlock detection
    [ ]   Built-in scriptable tester for rt-mutexes
    [ ]   Spinlock and rw-lock debugging: basic checks
    [ ]   Mutex debugging: basic checks
    [ ]   RW-sem debugging: basic checks
    [ ]   Spinlock debugging: sleep-inside-spinlock checking
    [ ]   Locking API boot-time self-tests
    [ ]   kobject debugging
    [*]   Compile the kernel with debug info
    [ ]   Debug VM
    v(+)

                 <Select>    < Exit >    < Help >
```

8. Select **Exit** to return to the main configuration menu.

Figure 9.30 General Setup Standard Kernel Features Options



```
       Configure standard kernel features (for small systems)
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    --- Configure standard kernel features (for small systems)
    [*]   Enable 16-bit UID system calls (NEW)
    [*]   Sysctl syscall support (NEW)
    [*]   Load all symbols for debugging/ksymoops
    [*]     Include all symbols in kallsyms
    [ ]     Do an extra kallsyms pass
    [*]   Support for hot-pluggable devices (NEW)
    [*]   Enable support for printk (NEW)
    [*]   BUG() support (NEW)
    [*]   Enable ELF core dumps (NEW)
    [*]   Enable full-sized data structures for core (NEW)
    v(+)

                 <Select>    < Exit >    < Help >
```

9. Select **Exit** to exit the configuration screen.

10. Select **Yes** when asked if you want to save your configuration.

    The kernel build begins and proceeds to completion.

The *LTIBInstallDir*`/rpm/BUILD/linux` directory contains `vmlinux`, the uncompressed kernel image with debug symbols.

Other files located in the *LTIBInstallDir*`/rpm/BUILD/linux` directory are:

`uImage` - the compressed kernel image without debug symbols and with a special header used by `u-boot`.

`vmlinux.bin` - the uncompressed kernel image with no debug symbols.

Creating CodeWarrior Project

1. Start the CodeWarrior IDE.
2. Select **File > Import**.

   The **Import** wizard appears.
3. Expand the **C/C++** group and select **MCU Executable**.
4. Click **Next**.
5. Specify a name in the **New Project Name** text box.
6. Click **Next**.

   In the **File to import** text box, specify the path specify the path of the kernel file `(vmlinux)` compiled with debug info.
7. Select ColdFire Linux/uClinux kernel image.
8. Select derivative
9. Click **Next**.
10. Select the connection.
11. Click **Finish**.

    The wizard will create the specified launch configuration.

You may edit the launch configuration and specify the boot parameters, like Command Line Settings or RAM Disk and Residual Data. Also, for MMU-enabled systems, check the Enable Memory Translation checkbox and set the proper Physical and Virtual Base addresses and memory size.

The **Residual Data** panel is filled with the information obtained by issuing the `bdinfo` command in u-boot. This represents the values of the `bdinfo` structure in `LTIB InstallDir/rpm/BUILD/linux/include/asm`.

12. Click the **Source** tab.
13. To enter the source mapping configuration, click **Add**.

    The **Add Source** dialog box appears.
14. Select **Path Mapping**.
15. Click **OK**.

    A new path mapping container appears in the **Source Lookup Path** area.

16. Select **Path Mapping: New Mapping** and click **Edit**.

    The **Path Mappings** dialog box appears.

17. Click **Add**.

    The **Path Mapping** dialog box appears.

18. Match the compilation path; the actual path where the Linux kernel was built with the paths where you have copied them - if this is the case.

10

Connections — HCS08

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the HCS08 simulator or the target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the Launch Configuration Settings panel. Launch Configuration Settings panel can be accessed by clicking on the Edit button located within the Main tab of the Debug Configurations dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with simulation platforms and hardware devices that are part of the HCS08 device family.

The topics in this chapter are:

- Changing Connection in IDE
- P&E Full Chip Simulation
- P&E HCS08 Multilink\Cyclone Pro\OSBDM

Changing Connection in IDE

Full Chip Simulation (FCS) connection runs a complete simulation of all processor peripherals and I/O on your personal computer. Thus, when debugging an FCS project for a selected derivative it is not necessary to connect your PC with a Microcontrollers development or target board.

To select Full Chip Simulation as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Device and Connection** page appears.

4. Expand the HCS08 tree control and select the derivative or board you would like to use. For example, select `HCS08 > HCS08D Family > 9S08DE32`.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Full Chip Simulation** checkbox.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

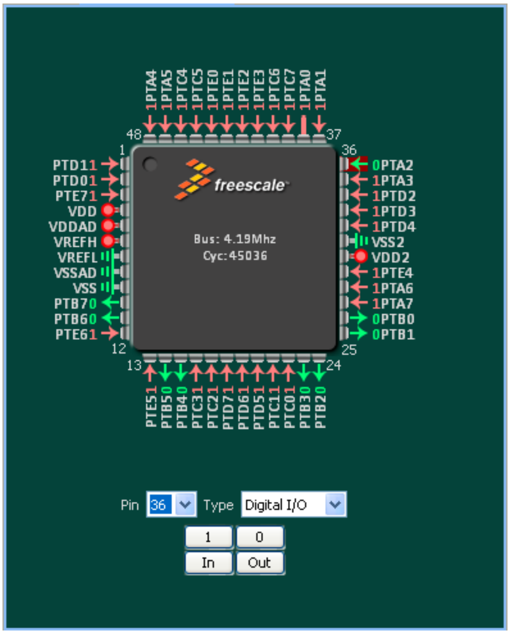
P&E Full Chip Simulation

This topic explains Chip View, which is a time-saving FCS feature. Additionally, it describes the settings of the connections that interface the CodeWarrior debugger with the HCS08 simulator.

Chip View

Chip View is an innovative feature designed to simplify Full Chip Simulation (FCS) and In-Circuit Debugging (ICD) sessions. The **Chip View** provides instantaneous access to internal modules of the chip and lets you instantly change any of the pin properties by clicking the pins. Each pin features the current pin direction, input/output value, and the name of the signal that reflects the current module that controls it. These data features are updated every 50ms throughout a running FCS or ICD session.

Figure 10.1 Chip View


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
freescale
Bus: 4.19Mhz
Cyc:45036
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

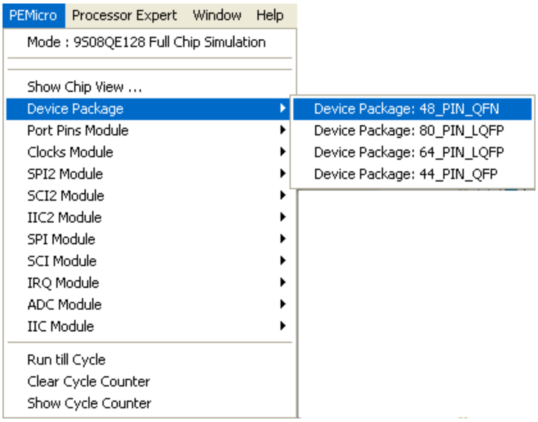
Pin 36 Type Digital I/O

To open **Chip View**, perform these steps.

1. From the IDE menu bar, select **PEMicro > Device Package > Device Package:<** *Pin*>, where < *Pin*> is the pin package you would like to work with. (Refer Figure 10.2).

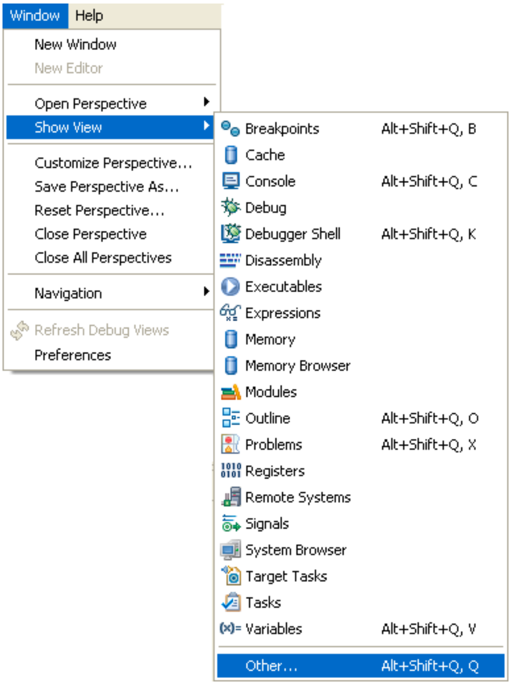
   The **Device Package** can be changed before or after the Chip View window is invoked within the CodeWarrior IDE.

Figure 10.2 Device Package Extended Menu


PEMicro
Processor Expert
Window
Help
Mode : 9S08QE128 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
SPI2 Module
SCI2 Module
IIC2 Module
SPI Module
SCI Module
IRQ Module
ADC Module
IIC Module
Run till Cycle
Clear Cycle Counter
Show Cycle Counter
Device Package: 48_PIN_QFN
Device Package: 80_PIN_LQFP
Device Package: 64_PIN_LQFP
Device Package: 44_PIN_QFP

2. From the IDE menu bar, select **Window > Show View > Others** (Figure 10.3).

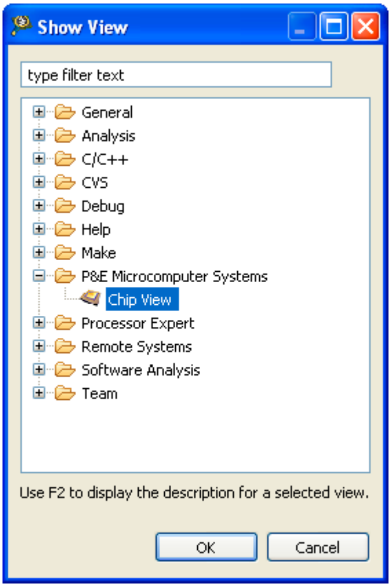
Figure 10.3 Show View Extended Menu


Window
Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Refresh Debug Views
Preferences
Breakpoints Alt+Shift+Q, B
Cache
Console Alt+Shift+Q, C
Debug
Debugger Shell Alt+Shift+Q, K
Disassembly
Executables
Expressions
Memory
Memory Browser
Modules
Outline Alt+Shift+Q, O
Problems Alt+Shift+Q, X
Registers
Remote Systems
Signals
System Browser
Target Tasks
Tasks
Variables Alt+Shift+Q, V
Other... Alt+Shift+Q, Q

The **Show View** dialog box appears.

3. Expand **P&E Microcomputer Systems** and select **Chip View** (Figure 10.4).

Figure 10.4 Show View Menu


Show View
type filter text
General
Analysis
C/C++
CVS
Debug
Help
Make
P&E Microcomputer Systems
Chip View
Processor Expert
Remote Systems
Software Analysis
Team

Use F2 to display the description for a selected view.

OK
Cancel

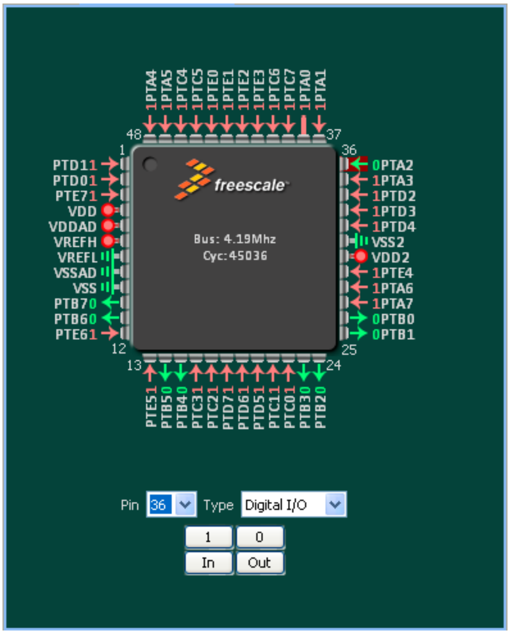
4. To change direction and values of the pin, double-click the corresponding arrow or the number value. Details are listed below.

NOTE If you close the **Chip View** Window during debug session, the Chip View will not be accessible. You must reopen the **Chip View** window and restart the current debugging session and to open the **Chip View** window again. Closing **Chip View** should slightly improve the performance during existing debug session.

Chip GUI - Ports Module Support

You can change the pin’s direction and values by double-clicking the corresponding arrow or the number value. Figure 10.5 is an example of what the **Chip View** may look like before any changes are made. When the pin direction is input, the pin will display the current pin input value. When the pin direction is output, you have the option of double-clicking the number value to control the output value for the pin.

Figure 10.5 Chip View Display Before Change

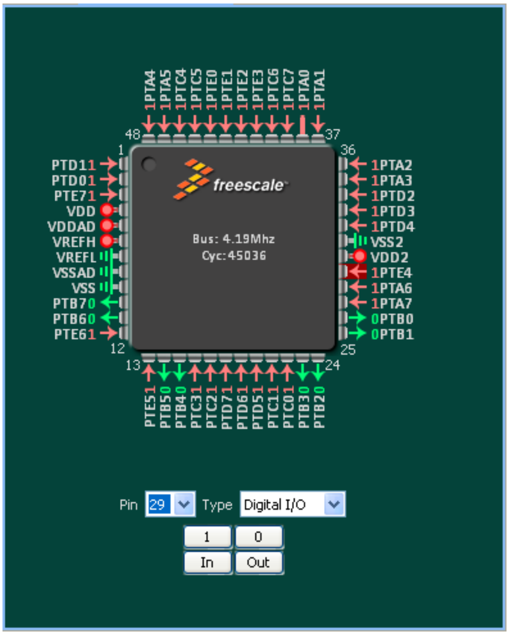

1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
freescale
1
0
In
Out

Bus: 4.19Mhz
Cyc:45036

Pin 36 Type Digital I/O

Figure 10.6 is an example of the PTA2 pin value being changed from 0 to 1 by double-clicking on the number value.

Figure 10.6 Chip View Display After Change



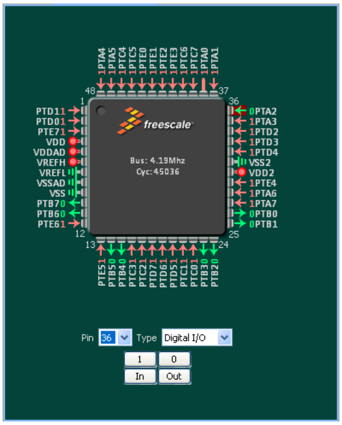
Bus: 4.19Mhz
Cyc:45036

Pin 29 Type Digital I/O

1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
1PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
freescale
1
0
In
Out

When you double-click the pin's value or the direction, a pin configuration dialog box appears beneath the Chip View diagram (Figure 10.7). In the pin configuration options, you can change the I/O settings for a given pin. You can select a pin from the pin-number drop-down list, select between analog and digital signals, and switch the pin directions. For the digital I/O signal, you can switch between high or low signals (Figure 10.7).

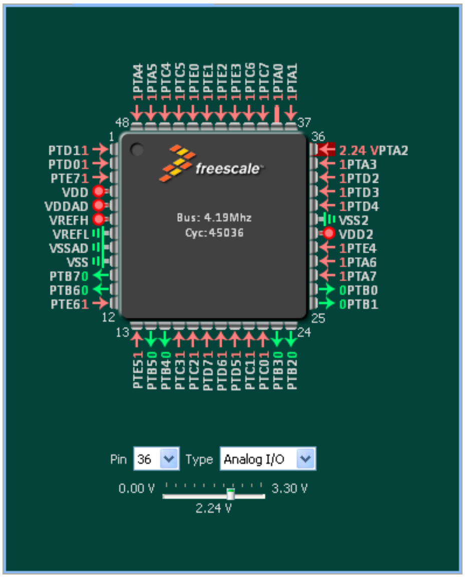
Figure 10.7 Chip View with Digital Pin Configuration Options


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
freescale
Bus: 4.19Mhz
Cyc:45036
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

Pin 36 Type Digital I/O

For the analog input signal, you can use the slider to change the analog signal value (Figure 10.8).

Figure 10.8 Chip View with Analog Pin Configuration Options


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
2.24 VPTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
freescale
Bus: 4.19Mhz
Cyc:45036
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
0.00 V
3.30 V

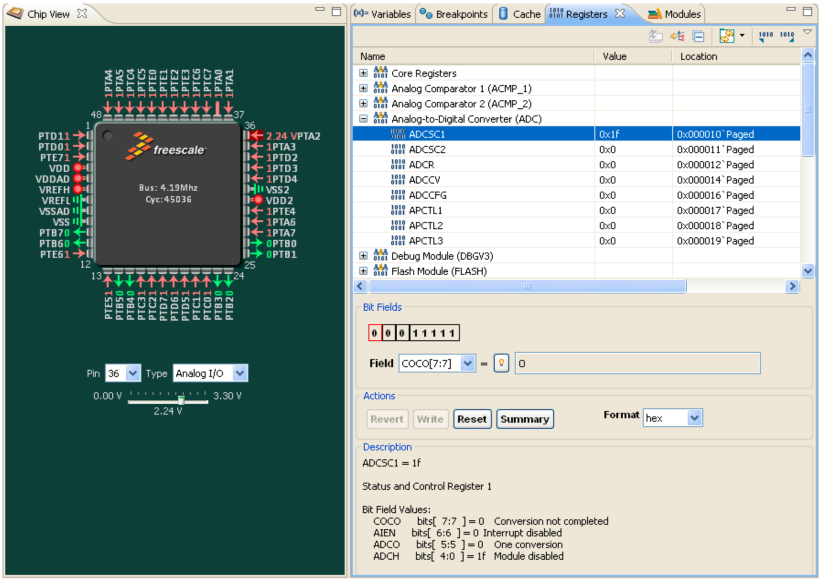
Pin 36 Type Analog I/O

2.24 V

Chip GUI - Analog to Digital Module Support

The Analog to Digital (ATD) Module has a higher priority than the General Pin I/O module. Therefore, you have an ATD channel enabled and the ATD input buffer is empty, current input value on a pin will be converted and displayed in the ATD data conversion register (Figure 10.9).

Figure 10.9 Chip View with ATD


Chip View
1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
2.24 VPTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
freescale
Bus: 4.19Mhz
Cyc:45036
0.00 V
3.30 V
2.24 V
Variables
Breakpoints
Cache
Registers
Modules
Bit Fields
0 0 0 1 1 1 1 1
Actions
Revert
Write
Reset
Summary
Description

Pin 36 Type Analog I/O

Name	Value	Location
Core Registers		
Analog Comparator 1 (ACMP_1)		
Analog Comparator 2 (ACMP_2)		
Analog-to-Digital Converter (ADC)		
ADCSC1	0x1f	0x000010`Paged
ADCSC2	0x0	0x000011`Paged
ADCR	0x0	0x000012`Paged
ADCCV	0x0	0x000014`Paged
ADCCFG	0x0	0x000016`Paged
APCTL1	0x0	0x000017`Paged
APCTL2	0x0	0x000018`Paged
APCTL3	0x0	0x000019`Paged
Debug Module (DBGV3)		
Flash Module (FLASH)		

Field COCO[7:7] = 0

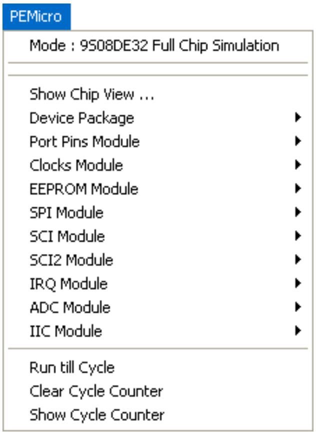
Format hex

```
ADCSC1 = 1f

Status and Control Register 1

Bit Field Values:
   COCO     bits[  7:7  ] = 0    Conversion not completed
   AIEN    bits[  6:6  ] = 0  Interrupt disabled
   ADCO     bits[  5:5  ] = 0    One conversion
   ADCH     bits[  4:0  ] = 1f   Module disabled
```

Module Options

The PEMicro menu (Figure 10.10) includes the Full Chip Simulation options for the modules that have specialty commands associated with them for a chosen device.

Figure 10.10 PEMicro Menu


PEMicro
Mode : 9S08DE32 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
EEPROM Module
SPI Module
SCI Module
SCI2 Module
IRQ Module
ADC Module
IIC Module
Run till Cycle
Clear Cycle Counter
Show Cycle Counter

The options available are:

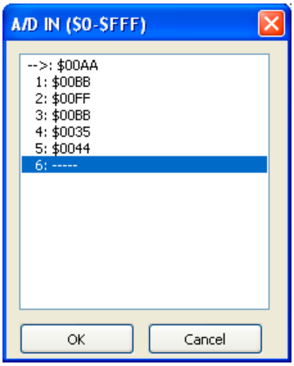
- Analog-to-Digital Converter Module
- 16-Bit Analog-to-Digital Converter Module
- Clock Generation Module
- Digital-to-Analog Converter Module
- EEPROM Module
- Fault Detection and Shutdown Module
- Flash Module
- Flextimer Module
- High-Speed Analog Comparator Module
- Inter-Integrated Circuit Module
- Interrupt Priority Controller Module
- External Interrupt (IRQ) Module
- Keyboard Interrupt Module
- Liquid Crystal Display Driver Module
- Modulo Timer Interrupt Module
- MSCAN Controller Module
- Programmable Delay Block Module
- Programmable Gain Amplifier Module
- Programmable Reference Analog Comparator Module

- Input/Output (I/O) Ports Module
- Serial Communications Interface Module
- Slave LIN Interface Controller (SLIC) Module
- Serial Peripheral Interface Module
- Timer Interface Module
- Time Of Day Module Option
- Universal Serial Bus (USB) Module
- Voltage Reference Module

Analog-to-Digital Converter Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Analog-to-Digital Conversion (ADC) module, including data input on all ADC channels, flag polling, interrupt operation, and the bus and CGMXCLK reference clock sources. FCS mode uses the buffered input structure to simulate the ADC inputs. You can queue up to 256 data values. To queue the ADC Input Data, use the `ADDI` command in the command prompt. If the data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used as input to the ADC. The conversion takes place after a proper value is written to the ADC Status and Control register. Once the conversion occurs, the arrow moves to the next value in the ADC Buffer.

Figure 10.11 ADC IN Buffer Display

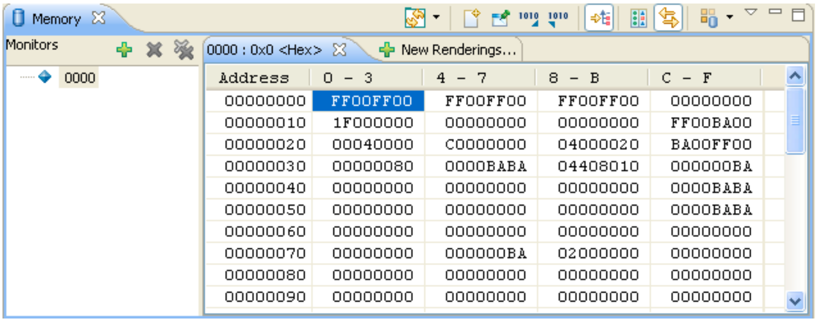


```
-->: $00AA
  1: $00BB
  2: $00FF
  3: $00BB
  4: $0035
  5: $0044
  6: -----
```

The `ADCLR` command can be used at any point to flush the input buffer for the ADC simulation.

After the conversion is complete, the first queued value is passed from the data buffer into the ADC data register. It can be observed in the Memory window by displaying the memory location corresponding to the ADC data register.

Figure 10.12 Memory Component Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

When the conversion is complete, FCS sets the appropriate flag. If interrupts are enabled, the Program Counter changes flow to the interrupt routine (as defined in the vector space of the MCU).

NOTE For more information on ADC configuration, refer to the Freescale user manual for your microprocessor.

ADC Module Commands

The following commands are available for the HC08/HCS08 ADC Module.

ADDI Command

The `ADDI` command lets you input the data into the ADC converter. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used by the ADC. The maximum number of input values is 256 bytes.

Syntax

```
>gdi ADDI [<n>]
```

Where:

`<n>` The value to be entered into the next location in the input buffer.

Example

```
>gdi ADDI $55
```

Set the next input value to the ADDI to $55

```
>gdi ADDI
```

Pull up the data window with all the input values.

ADCLR Command

Use the `ADCLR` command to flush the input buffer for ADC simulation. This resets the input data buffer and clears out all values. Notice that if the ADC is currently using a value, this command does not prevent the ADC from using it.

NOTE Refer the `ADDI` command for information on how to access the input buffer of the ADC interface.

Syntax

```
>gdi ADCLR
```

Example

```
>gdi ADCLR
```

Clear the input buffer for ADC simulation.

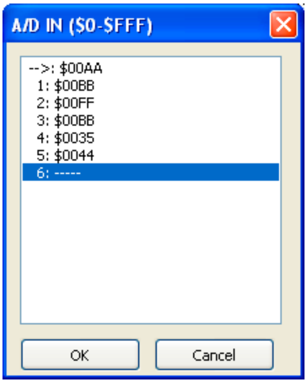
16-Bit Analog-to-Digital Converter Module

The following commands are available for the HCS08 ADC16V1 Module.

ADDI Command

The ADDI command allows the user to input the data into the ADC16 converter. If a data parameter is given, the value is placed into the next slot in the input buffer. If no data parameter is given, a window is displayed with the input buffer values shown in Figure 10.13. Input values can be entered while the window is open. An arrow points to the next value that will be used by the ADC16. The maximum number of input values is 256 bytes.

Figure 10.13 ADC16 IN Buffer Display



```
-->: $00AA
  1: $00BB
  2: $00FF
  3: $00BB
  4: $0035
  5: $0044
  6: -----
```

After the conversion is complete, the first queued value is passed from the data buffer into the ADC16 data register. It can be observed in the memory window by displaying the memory location corresponding to the ADC16 data register.

Syntax

```
>gdi ADDI <n>
```

Where:

`<n>` The value to be entered into the next location in the input buffer.

Example

```
>gdi ADDI $55
```

Set the next ADDI input value to $55

```
>gdi ADDI
```

Pull up the data window with all the input values.

ADCLR Command

Use the ADCLR command to flush the input buffer for ADC16 simulation. This resets the input data buffer and clears out all values. Note that if the ADC16 is currently using a value, this command does not prevent the ADC16 from using it. Refer the ADDI command for information on how to access the input buffer of the ADC16 interface.

Syntax

```
>gdi ADCLR
```

Example

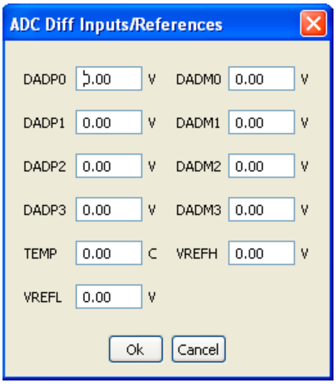
```
>gdi ADCLR
```

Clear the input buffer for ADC simulation.

ADDID Command

The ADDID command allows the user to input the differential data into the ADC16 converter. A window is displayed that allows the user to input differential values, as shown in Figure 10.14. The user may specify the differential data of the Differential Analog Channel (DADx) input voltages, the Temperature Sensor (TEMP) input, and the Voltage References (VREF).

Figure 10.14 ADC Differential Data Display



DADP0	0.00	V	DADM0	0.00	V
DADP1	0.00	V	DADM1	0.00	V
DADP2	0.00	V	DADM2	0.00	V
DADP3	0.00	V	DADM3	0.00	V
TEMP	0.00	C	VREFH	0.00	V
VREFL	0.00	V			

When using FCS, the ADDID command shows the simulated differential inputs to the ADC16 module.

Syntax

```
>gdi ADDID
```

Example

```
>gdi ADDID
```

Pull up the differential data window

Clock Generation Module

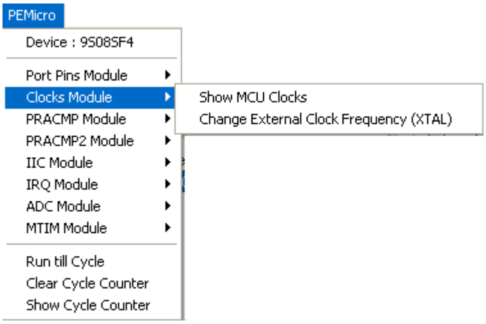
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Clock Generation Module (ICG), including:

- Phase Locked Loop (PLL) generation
- Automatic lock detection

- Interrupt
- Acquisition
- Tracking
- Flag polling

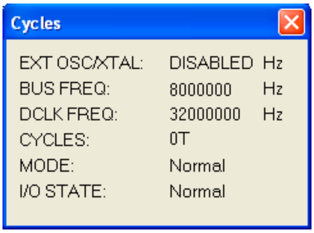
FCS mode uses a simulated External Oscillator Frequency change command (`XTAL`) lets you input the desired XTAL value. To check the current value of the External Oscillator, Bus Frequency and CGMXCLK Frequency, open the HCS08FCS menu and select Clocks Module > Show MCU Clocks.

Figure 10.15 Clocks Module Extended Menu


PEMicro
Device : 9S08SF4
Port Pins Module
Clocks Module
PRACMP Module
PRACMP2 Module
IIC Module
IRQ Module
ADC Module
MTIM Module
Run till Cycle
Clear Cycle Counter
Show Cycle Counter
Show MCU Clocks
Change External Clock Frequency (XTAL)

Once you select the MCU Clocks menu, the Cycles window displays all of the aforementioned Clock Frequencies, or you can select the Show Cycle Counter option within the FCS menu to get the same window.

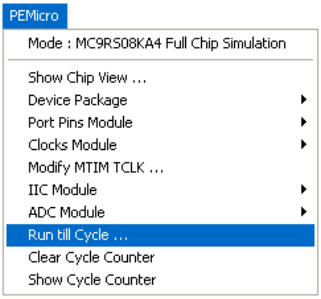
Figure 10.16 Frequency Display



```
EXT OSC/XTAL:   DISABLED  Hz
BUS FREQ:       8000000   Hz
DCLK FREQ:      32000000  Hz
CYCLES:         0T
MODE:           Normal
I/O STATE:      Normal
```

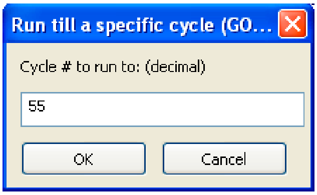
Within the FCS menu, you can select the Run till Cycle option, which lets you begin code execution and stop execution when the specified cycle count is reached. Note that the parameter given is not the number of cycles that executed, but rather the total cycle-count of the simulator (displayed in the Register Window).

Figure 10.17 Run till Cycle Command


PEMicro
Mode : MC9RS08KA4 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
Modify MTIM TCLK ...
IIC Module
ADC Module
Run till Cycle ...
Clear Cycle Counter
Show Cycle Counter

This command is extremely useful for verifying specific timings of a given event, running until a given event is complete, or just before it completes to enable stepping through the event or any application where cycle-timed execution is desired.

Figure 10.18 Run till a specific cycle Dialog Box

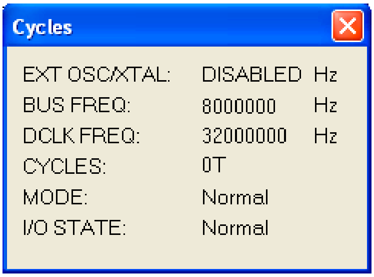

Run till a specific cycle (GO...
OK
Cancel

Cycle # to run to: (decimal)

55

You can also select the Clear Cycle Counter option within the FCS menu, which clears the cycle counter. If you select the Show Cycle Counter option within the FCS menu, you can check to make sure that the cycle counter is zero.

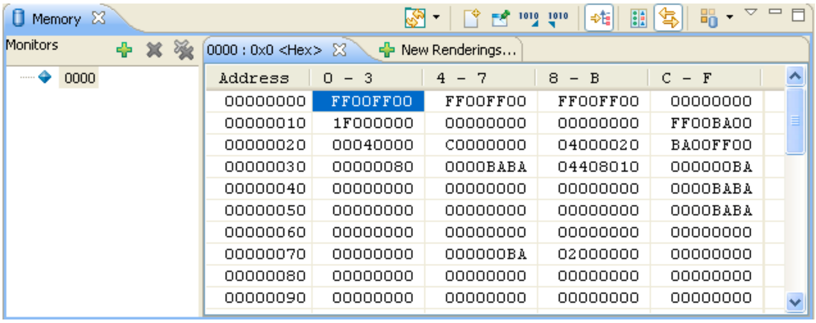
Figure 10.19 Cycles Dialog Box with Cleared Counter



EXT OSC/XTAL:	DISABLED	Hz
BUS FREQ:	8000000	Hz
DCLK FREQ:	32000000	Hz
CYCLES:	0T	
MODE:	Normal	
I/O STATE:	Normal	

Once the ICG is properly configured, you can monitor the status of the PLL by polling the corresponding flag. If PLL interrupt is enabled, FCS jumps to an appropriate subroutine, as long as the interrupt vector is properly defined. To observe the flag going up as a result of the corresponding CPU event, situate your Memory window on the memory location of the ICG Status and Control register.

Figure 10.20 Memory Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

For more information on how to properly configure Clock Generation, refer to the Freescale reference manual for your microprocessor.

Clock Generation Module Commands

The following commands are available for the HC08/HCS08 Clock Generation Module.

XTAL Command

Use the `XTAL` command to change the value of the simulated external oscillator. This in turn affects the input to the PLL/DCO, and therefore the bus frequency. The P&E simulator is a cycle-based simulator, so changing the XTAL value does not affect the speed of simulation. It does, however, affect the ratio in which peripherals receive cycles. Certain peripherals that run directly from the XTAL will run at different speeds than those that run from the bus clock.

Syntax

```
>gdi XTAL <n>
```

where, `<n>`, by default, is a hexadecimal number, representing the simulated frequency of an external oscillator. Adding the suffix `'t'` to the `'n'` parameter forces the input value to be interpreted as base 10.

Example

```
>gdi XTAL
```

Brings up an input window. The default base for this input value is 10. However, this value can be forced to a hexadecimal format through use of the suffix `'h'`.

Digital-to-Analog Converter Module

In Full Chip Simulation (FCS) Mode, this module lets you simulate all the functionality of the 5-bit Digital-to-Analog Converter (DAC) module. This module provides 32 distinct selectable voltage levels through the use of a 32-tap resistor ladder network and a 32-to-1 multiplexer. Each DAC module output can be routed to an HSCMP input.

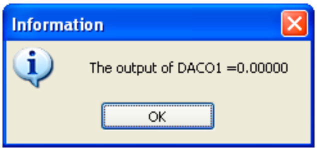
Digital-to-Analog User Commands

The following DAC commands are available for the HCS08.

SHOWDACO1 Command

The SHOWDACO1 command displays the DAC Output dialog box shown in Figure 10.21.

Figure 10.21 Figure DAC Output Dialog Box



The output of DACO1 =0.00000

Syntax

```
>gdi SHOWDACO1
```

Example

```
>gdi SHOWDACO1
```

Show DAC Output Dialog Box

EEPROM Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of EEPROM module including sector erase abort, burst programming capability, security feature, flexible block protection and vector redirection, and command interface for fast program and erase operation.

EEPROM User Commands

The following `EEPROM` commands are available for the HCS08.

EEPROM<x> Command

The `EEPROM<x>` command simulates changing of the EEPROM page for devices that have paged EEPROM.

Syntax

```
>gdi EEPROM<x>
```

Where:

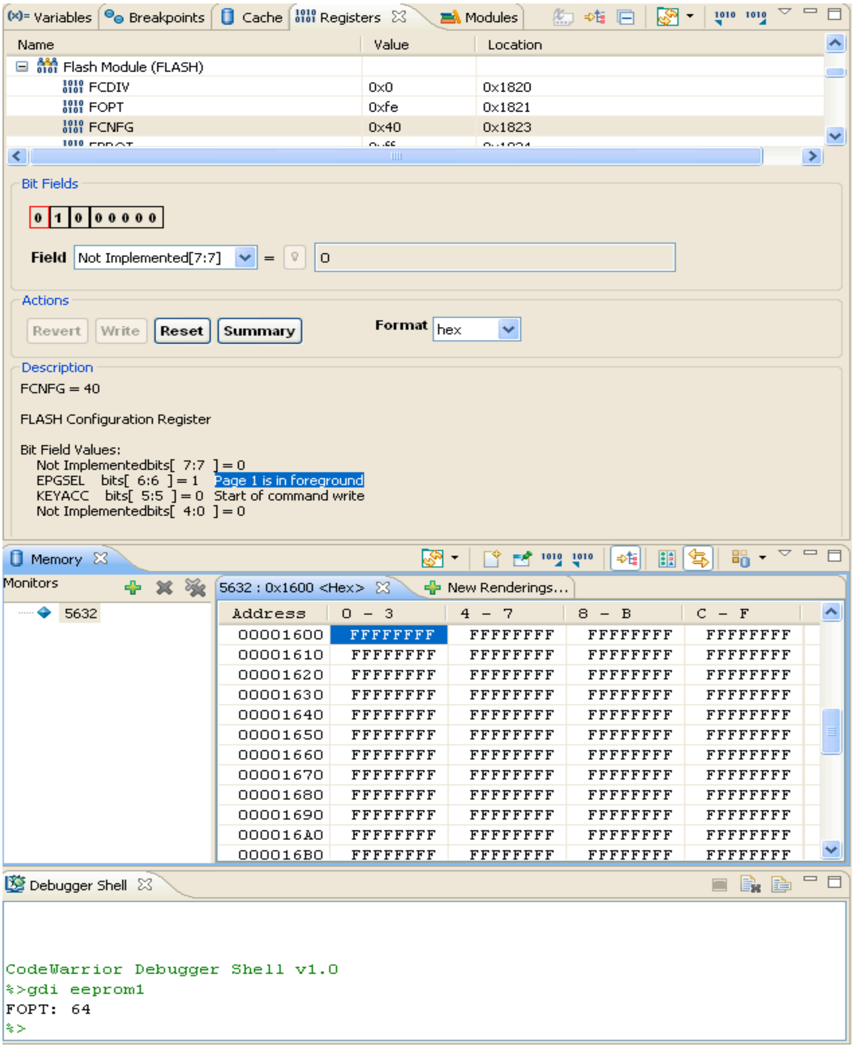
`<x>` is the letter representing corresponding EEPROM page number

Example

```
> gdi EEPROM1
```

Simulate change to EEPROM page 1.

Figure 10.22 Example of using EEPROM<x>Command


Variables
Breakpoints
Cache
Registers
Modules
Bit Fields
Actions
Revert
Write
Reset
Summary
Description
Memory
Monitors
New Renderings...
Debugger Shell

Name	Value	Location
Flash Module (FLASH)		
FCDIV	0x0	0x1820
FOPT	0xfe	0x1821
FCNFG	0x40	0x1823

0 1 0 0 0 0 0 0

Field Not Implemented[7:7] = 0

Format hex

```
FCNFG = 40

FLASH Configuration Register

Bit Field Values:
   Not Implementedbits[  7:7  ] = 0
   EPGSEL    bits[  6:6  ] = 1   Page 1 is in foreground
   KEYACC    bits[  5:5  ] = 0   Start of command write
   Not Implementedbits[  4:0  ] = 0
```

5632

5632 : 0x1600 <Hex>

```
Address    0 - 3     4 - 7     8 - B     C - F
00001600   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001610   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001620   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001630   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001640   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001650   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001660   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001670   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001680   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
00001690   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000016A0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
000016B0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
```

```
CodeWarrior Debugger Shell v1.0
%>gdi eeprom1
FOPT: 64
%>
```

Fault Detection and Shutdown Module

In Full Chip Simulation (FCS) Mode, this module lets you simulate all the functionality of the Fault Detection and Shutdown (FDS) module. When a fault condition occurs, the module provides a mechanism to immediately place port pins into a pre-defined state; the output pin of FDS can be configured as output 0, output 1, high impedance, or bypass during shutdown. The module can configure up to 8 fault input sources and control up to 8 port pins.

Flash Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of Flash module including sector erase abort, burst programming capability, security feature, flexible block protection and vector redirection, and command interface for fast program and erase operation.

Flash User Commands

The following Flash commands are available for the HCS08.

PPAGE <x> Command

The `PPAGE <x>` command simulates changing of Flash PPAGE for devices that have paged FLASH.

Syntax

```
>gdi PPAGE <x>
```

Where:

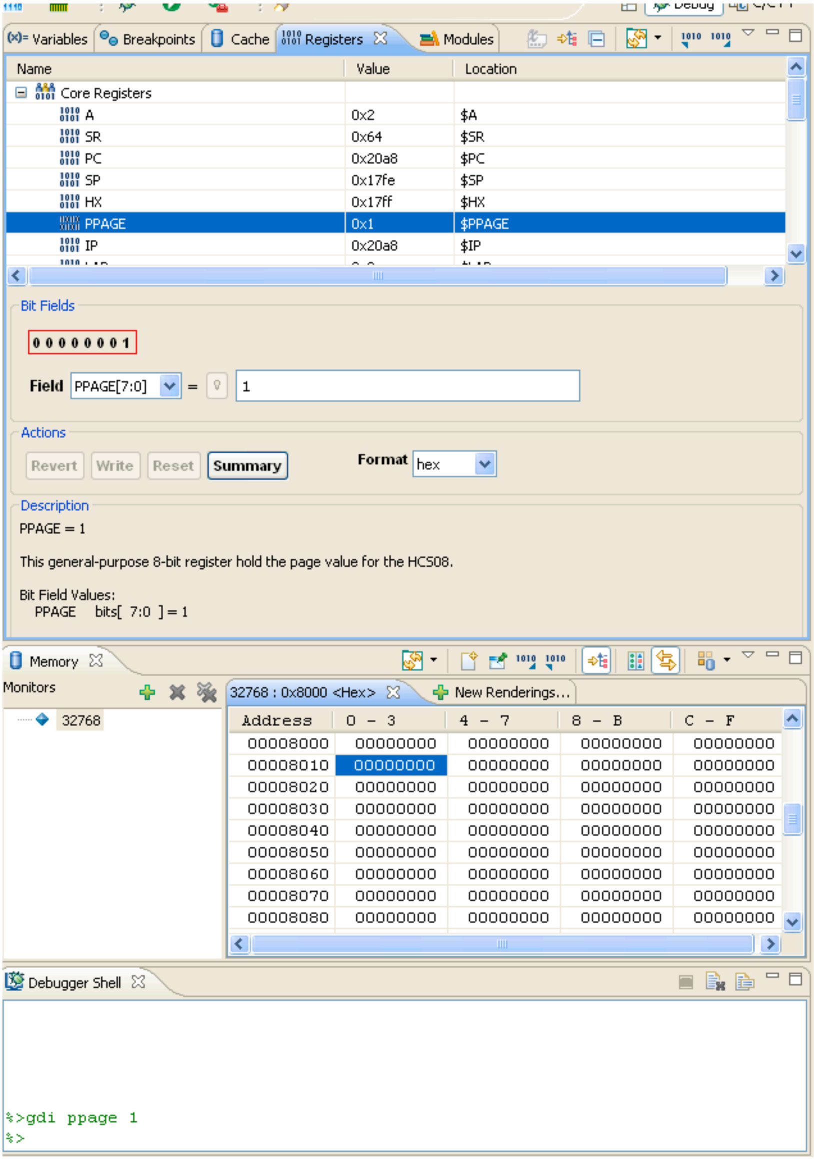
<x> is the letter representing corresponding PPAGE number

Example

```
>gdi PPAGE 1
```

Simulate change to PPAGE 1.

Figure 10.23 Example of using PPAGE <x> command


Variables
Breakpoints
Cache
Registers
Modules
Bit Fields
Field
Actions
Revert
Write
Reset
Summary
Format
Description
Memory
Monitors
New Renderings...
Debugger Shell

Name	Value	Location
Core Registers		
A	0x2	$A
SR	0x64	$SR
PC	0x20a8	$PC
SP	0x17fe	$SP
HX	0x17ff	$HX
PPAGE	0x1	$PPAGE
IP	0x20a8	$IP

0 0 0 0 0 0 0 1

Field PPAGE[7:0] = 1

Format hex

PPAGE = 1

This general-purpose 8-bit register hold the page value for the HCS08.

Bit Field Values:
PPAGE   bits[7:0] = 1

32768

32768 : 0x8000 <Hex>

Address	0 - 3	4 - 7	8 - B	C - F
00008000	00000000	00000000	00000000	00000000
00008010	00000000	00000000	00000000	00000000
00008020	00000000	00000000	00000000	00000000
00008030	00000000	00000000	00000000	00000000
00008040	00000000	00000000	00000000	00000000
00008050	00000000	00000000	00000000	00000000
00008060	00000000	00000000	00000000	00000000
00008070	00000000	00000000	00000000	00000000
00008080	00000000	00000000	00000000	00000000

```
%>gdi ppage 1
%>
```

Flextimer Module

In Full Chip Simulation (FCS) Mode, this option lets you simulate all the functionality of the Flextimer (FTMV2) module, including:

- Input capture/output compare
- Pulse width modulation
- Internal or external clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation.
- All channels can be configured for center-aligned PWM mode
- Each pair of channels can be combined to generate a PWM signal (with independent control of both edges of PWM signal)
- The FTM channels can operate as pairs with equal outputs, pairs with complimentary outputs, or independent channels (with independent outputs)
- The dead-time insertion is available for each complementary pair
- Generation of triggers (hardware trigger)

High-Speed Analog Comparator Module

In Full Chip Simulation (FCS) mode, this option lets you simulate all the functionality of the High-Speed Analog Comparator (HSCMP) module, including data input on all HSCMP channels, flag polling, and interrupt operation, as well as output connection to PDB input triggers. The user can utilize either the HSCMP Inputs display form or command-line commands to provide inputs to the HSCMP module.

High-Speed Analog Comparator User Commands

The following commands are available for the HCS08 HSCMP Module.

HSC<x>INPUT<y> Command

The HSC<x>INPUT<y> command lets you input a voltage value for the external analog input CIN<y>. For HCS08 devices that have more than one HSCMP module, <x> is the number representing the corresponding module. For HCS08 devices that only have one HSCMP module, <x>=1.

Syntax

```
>gdi HSC<x>INPUT<y> <n>
```

Where:

<x> is the number representing the corresponding HSCMP module

<y> is the number representing the corresponding external analog input CIN

<n> is the voltage input value

Examples

```
>gdi HSC1INPUT1 5.00
```

Simulate a 5.00 volt input on CIN1 of HSCMP1.

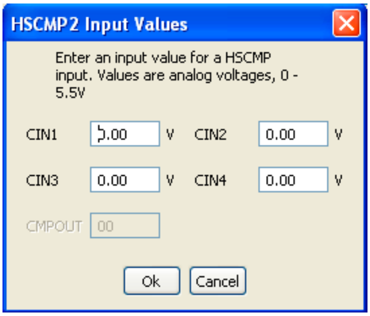
```
>gdi HSC4INPUT3 2.50
```

Simulate a 2.50 volt input on CIN3 of HSCMP4.

HSC<x>INPUTS Command

In FCS mode, the HSC<x>INPUTS command opens the HSCMP<x> Input Value dialog box shown in Figure 10.24. The user may then use this box to specify the external analog input voltages. For HCS08 devices that have more than one HSCMP module, <x> is the number representing the corresponding module. For HCS08 devices that only have one HSCMP module, <x>=1.

Figure 10.24 HSCMPx Input Value Dialog Box



Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

CIN1 5.00 V CIN2 0.00 V

CIN3 0.00 V CIN4 0.00 V

CMPOUT 00

When using FCS, the HSC<x>INPUTS command shows the simulated input analog voltages to any applicable HSCMPx module.

Syntax

```
>gdi HSC<x>INPUTS
```

Where:

<x> is the number representing the corresponding HSCMP module

Example

```
>gdi HSC1INPUTS
```

Show input analog voltages.

Inter-Integrated Circuit Module

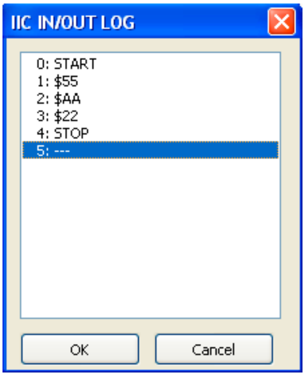
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Inter-Integrated Circuit (IIC) module including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Master and slave modes of operation
- START and STOP signal generation detection
- Acknowledge bit generation detection

FCS mode uses the buffered input/output structure to simulate IIC inputs. You can queue up to 256 data bytes into the input buffer. The output buffer of the USB module can also hold 256 output bytes. To queue the IIC Input Packets, use the `IICDI <...>` command in the command prompt. For a more detailed description of the command, refer to the IIC Commands section. If the IIC packet parameters are properly defined, the packet is placed into the next slot in the input buffer. Otherwise, if no parameters are provided, an IIC Input Buffer window is displayed.

You can enter different IIC packet parameters while the window is open, including START, STOP, ACK, NACK and data bytes. An arrow points to the next byte to be used as input to the IIC. The data from the IIC input buffer is written to the IIC module registers once the IIC module is turned on and properly configured for receiving data from an external IIC device. Once simulation of the data transmission is over, the arrow moves to the next value in the IIC Input Buffer.

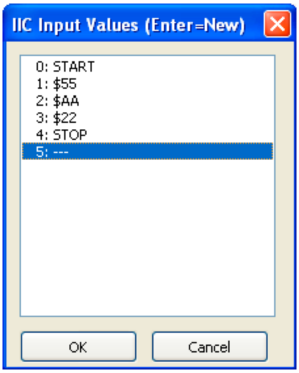
Figure 10.25 IIC Input Buffer Display



```
0: START
1: $55
2: $AA
3: $22
4: STOP
5: ---
```

The IIC data input/output log buffer simulation lets you gain access to the past 256 IIC data bytes that have been shifted in and out of the module. To bring up the IIC IN/OUT LOG buffer dialog box, use the `IICDO` command.

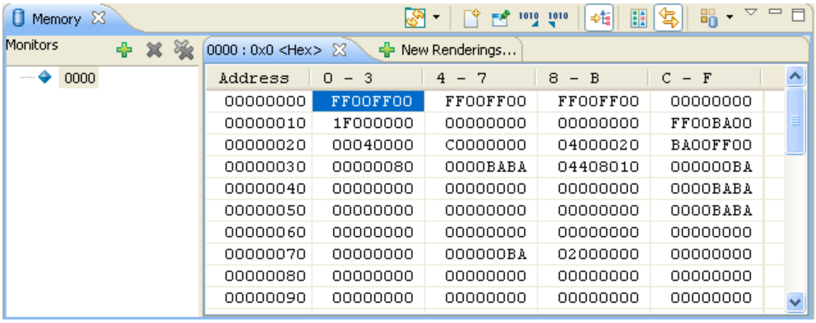
Figure 10.26 IIC IN/OUT LOG Buffer Display



```
0: START
1: $55
2: $AA
3: $22
4: STOP
5: ---
```

At any point, use the `IICCLR` command to flush the input as well as input/output log IIC buffers. After the IIC simulated input is received, the first queued-in data byte is passed from the data buffer into the corresponding IIC module registers. It can be observed in the Memory Window by displaying the appropriate register location there.

Figure 10.27 Memory Component Window



```
Address    0 - 3     4 - 7     8 - B     C - F
00000000   FF00FF00  FF00FF00  FF00FF00  00000000
00000010   1F000000  00000000  00000000  FF00BA00
00000020   00040000  C0000000  04000020  BA00FF00
00000030   00000080  0000BABA  04408010  000000BA
00000040   00000000  00000000  00000000  0000BABA
00000050   00000000  00000000  00000000  0000BABA
00000060   00000000  00000000  00000000  00000000
00000070   00000000  000000BA  02000000  00000000
00000080   00000000  00000000  00000000  00000000
00000090   00000000  00000000  00000000  00000000
```

You can also observe different IIC flags in the Memory window. If you run the module in Flag Polling mode, poll the flag corresponding to the expected IIC event. If the IIC interrupts are enabled, FCS jumps to an appropriate subroutine as long as the IIC interrupt vectors are properly defined.

NOTE For more information on how to configure IIC module for desired operation, refer to the Freescale user manual for your microprocessor.

Inter-Integrated Circuit Module Commands

The following commands are available for the HCS08 Inter-Integrated Circuit (IIC) module and the HC08 Multi-Master Inter-Integrated Circuit (MMIIC) module. Command function is identical even though the module names differ.

IICDI Command

The `IICDI` command lets you input data into a buffer of data to shift into the IIC module when it receives data from an external device. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. The maximum number of input values is 256. This command is useful for either inputting response data from a slave target or for inputting data packets from an external master. Note that when the microprocessor attempts to read an acknowledge from an external device, and the next value in the buffer is neither ACK nor NACK, the microprocessor automatically receives an ACK signal (i.e. assumes ACK unless NACK is specified).

Syntax

```
>gdi IICDI [<n>][START][STOP][ACK][NACK]
```

Where:

- `<n>` indicates the value to be entered into the next location in the input buffer
- `START` indicates the incoming START signal
- `STOP` indicates the incoming STOP signal
- `ACK` corresponds to ACK signal
- `NACK` corresponds to NACK signal

NOTE For a detailed description of the IIC protocol and a proper way to configure the IIC module, refer to the Freescale user manual for your microprocessor.

Example

```
>gdi IICDI
```

Pulls up the data window with all the input values

```
>gdi IICDI 22 33
```

This is an example of data being returned from a slave device. Once the MCU transmits a start signal and the target address, it receives an ACK from the slave device. An ACK is implied unless a NACK is specified via the `IICDI` command. The next two data bytes read are 22 and 23. If the microprocessor attempts to read another byte, it gets an $FF value followed by a NACK signal (NACK because nothing remains in the input buffer). The receiving device then generates a STOP signal. A more exact input from a device designed to return two bytes is:

```
>gdi IICDI ACK 22 ACK 23 NACK
```

IIC in master mode transmits to a slave:

- If the slave device acknowledges all output bytes of the transmitting device, there is no need to specify an input packet. If the master device is going to transmit an address and two bytes, the following packet is equivalent to no packet:

```
>gdi IICDI ACK ACK ACK
```

- If, however, the slave receiver is designed to generate a NACK signal after the second received data byte, the proper response packet is:

```
>gdi IICDI ACK ACK NACK
```

- The address result being the first ACK, the first data result being the second ACK, and the second data byte being the NACK.

IIC in MASTER mode is not acknowledged by any Slave:

```
>gdi IICDI NACK
```

- If the NACK signal is entered before the master device transmits a START signal, then the master device gets a NACK when it tries to read an acknowledge after the address is output. The master device then generates a STOP signal and releases the BUS.

IIC in SLAVE mode receives a Write from an external Master:

This example is for an external master that is writing to the microprocessor configured to simulate the slave mode operation. The packet contains both START and STOP signals which puts the simulated device into the slave mode.

```
>gdi IICDI START 55 AA 22 STOP
```

This input adds five values to the input queue, which is a packet from an external master, including the following procedure values:

- A start signal comes in
- The address $55 comes in specifying a write (slave receive). The Address Register in the current simulated device has been previously set to $55
- The data byte $AA comes in
- The data byte $22 comes in
- A STOP signal comes in

IICDO Command

The IICDO command displays a window, which shows data both shifted in and shifted out of the IIC peripheral. An arrow points to the last output value transmitted/received. The maximum number of output values that the buffer can hold is 256.

Syntax

```
>gdi IICDO
```

Example

```
>gdi IICDO
```

View data from the input/output log buffer for IIC simulation.

IICCLR Command

Use the `IICCLR` command to flush the input and output buffers for IIC simulation. This resets the buffers and clears all values. Notice that if the IIC is currently shifting a value, this command does not prevent the IIC from finishing the transfer.

Syntax

```
>gdi IICCLR
```

Example

```
>gdi IICCLR
```

Clear input and output buffers for IIC simulation.

Interrupt Priority Controller Module

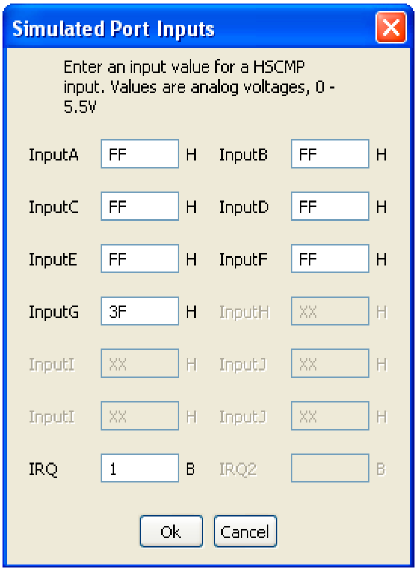
In Full Chip Simulation (FCS) Mode, this module simulates all the functionality of the Interrupt Priority Controller (IPC) module. This module provides a hardware-based, nested-interrupt mechanism in HCS08 MCUs and allows all prioritized non-software interrupts to interrupt. IPC features a four-level programmable interrupt priority for each source, supports prioritized preemptive interrupt service routines, and the interrupt priority mask can be modified during main flow or interrupt service execution. When the interrupt vector is being fetched, the module can automatically update the interrupt priority mask with its serviced interrupt source priority level and automatically store previous interrupt mask levels.

External Interrupt (IRQ) Module

In Full Chip Simulation (FCS) mode, this module simulates the input, flag polling and interrupt functionality of the External Interrupt (IRQ) module. FCS mode uses the `INPUTS` command and let you monitor and change the simulated value of the IRQ input pin state. Once you enter the `INPUTS` command into the command line prompt, the

Simulated Port Inputs window appears. Refer the `INPUT<x>` Command for more information about the various forms of this command. In addition, the state of the IRQ pin can be modified directly using the `IRQ<n>` command (documented below).

Figure 10.28 Simulated Port Inputs Dialog Box



Simulated Port Inputs

Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

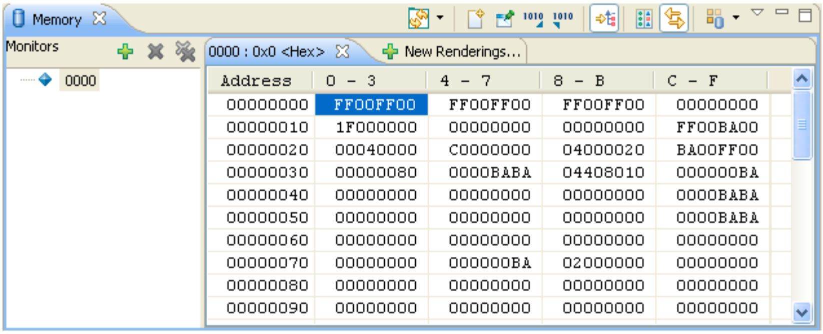
InputA FF H InputB FF H
InputC FF H InputD FF H
InputE FF H InputF FF H
InputG 3F H InputH XX H
InputI XX H InputJ XX H
InputI XX H InputJ XX H
IRQ 1 B IRQ2 B

An IRQ event occurrence sets the appropriate flag in the corresponding IRQ register. You can poll the IRQ flag if the Polling Mode is simulated. In the Interrupt Mode, the simulator branches to an appropriate interrupt subroutine as long as the IRQ interrupt vector is properly configured.

NOTE For more information on IRQ configuration, refer to the Freescale user manual for your microprocessor.

Following the IRQ event, you can observe the IRQ Flag being set in the IRQ Status and Control register.

Figure 10.29 Memory Component Window



Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

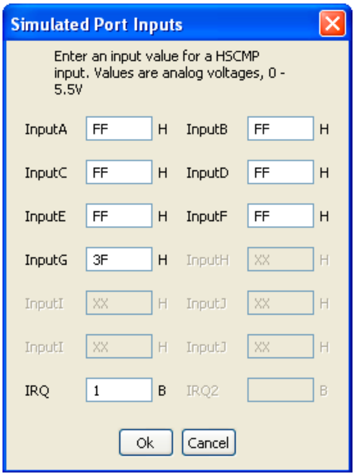
IRQ Commands

The following interrupt request command is available for the HC08/HCS08 processors.

INPUTS Command

In FCS and CPU-Only Simulation mode, the `INPUTS` command opens the Simulated Port Inputs dialog box shown in Figure 10.30. You may then use this box to specify the input states of port pins and IRQ.

Figure 10.30 Simulated Port Inputs Dialog Box



Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA	FF	H	InputB	FF	H
InputC	FF	H	InputD	FF	H
InputE	FF	H	InputF	FF	H
InputG	3F	H	InputH	XX	H
InputI	XX	H	InputJ	XX	H
InputI	XX	H	InputJ	XX	H
IRQ	1	B	IRQ2		B

When using In-Circuit Simulation mode, the `INPUTS` command shows the simulated input values for any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

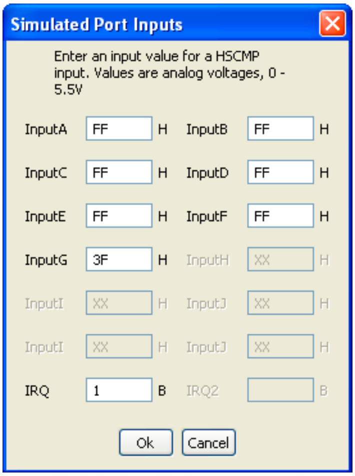
NOTE The IRQ pin state can be directly manipulated with the `IRQ` command. For example, IRQ 1 simulates a high state on the IRQ pin; likewise, IRQ 0 simulates a logic-low state on the IRQ pin.

Keyboard Interrupt Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Keyboard Interrupt (KBI) module, including the edge-only, edge and level interrupt, and flag polling modes of operation. FCS mode uses simulated port inputs to trigger the KBI event from the proper I/O port pin.

To define an input state of the specific port, write the `INPUT<x> <n>` command in the Command window. The `<x>` represents the corresponding I/O port, while `<n>` stands for the input value to write to this port. At the same time, you can use the `INPUTS` command to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Refer the documentation for Timer Module Commands for more information about the various forms of this command.

Figure 10.31 Simulated Port Inputs Dialog Box

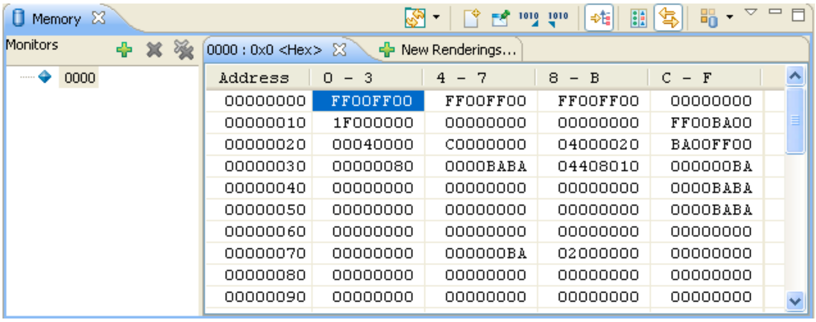


Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA	FF	H	InputB	FF	H
InputC	FF	H	InputD	FF	H
InputE	FF	H	InputF	FF	H
InputG	3F	H	InputH	XX	H
InputI	XX	H	InputJ	XX	H
InputI	XX	H	InputJ	XX	H
IRQ	1	B	IRQ2		B

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. To trigger the event, manipulate the inputs to the port in the appropriate manner, depending on whether the KBI is configured for edge-only or edge and level. Once the KBI event takes place, you can observe the KEYF Flag bit, which is a part of the Keyboard Status and Control register, in the Memory window.

Figure 10.32 Memory Component Window



Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

You can poll the KBI Interrupt Pending flag if the Polling Mode is simulated. In Interrupt Mode, the simulator branches to an appropriate interrupt subroutine as long as the KBI interrupt vector is properly configured.

NOTE For more information on KBI configuration, refer to the Freescale user manual for your microprocessor.

Keyboard Interrupt Commands

Use the following commands for Keyboard Interrupt manipulation.

INPUT<x> Command

The `INPUT<x>` command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

`<x>` is the letter representing corresponding port

`<n>` is an eight-bit simulated value for port <x>

Example

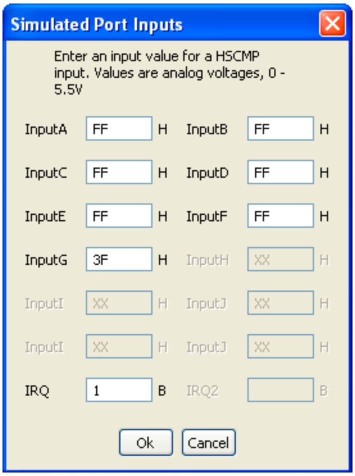
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation mode, the `INPUTS` command opens the Simulated Port Inputs dialog box shown in Figure 10.33. You may then use this box to specify the input states of port pins and IRQ.

Figure 10.33 Simulated Port Inputs Dialog Box



Simulated Port Inputs

Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA	FF	H	InputB	FF	H
InputC	FF	H	InputD	FF	H
InputE	FF	H	InputF	FF	H
InputG	3F	H	InputH	XX	H
InputI	XX	H	InputJ	XX	H
InputI	XX	H	InputJ	XX	H
IRQ	1	B	IRQ2		B

When using In-Circuit Simulation mode, the `INPUTS` command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

Liquid Crystal Display Driver Module

In Full Chip Simulation (FCS) mode, this option lets you simulate all the functionality of the Liquid Crystal Display (LCD) module, including programmable LCD frame frequency, front plane pin configuration, back plane pin configuration, programmable blink frequency, and LCD interrupt flag generation. By default, LCD front and back plane pins are mapped to match device use on the corresponding Freescale DEMO9S08xx device board.

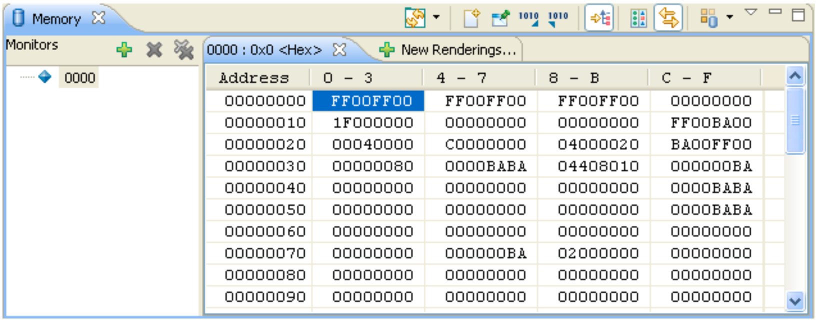
Modulo Timer Interrupt Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Modulo Timer Interrupt (MTIM) Module, including:

- Programmable MTIM clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation

Once the MTIM Status and Control register properly configures the operation of the module, the MTIM Counter starts incrementing. If modulo up count operation is enabled, you can observe the MTIM overflow flag in the MTIM Status and Control register in the Memory window.

Figure 10.34 Memory Component Window



Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

If the MTIM interrupt is enabled, the FCS jumps to an appropriate subroutine as long as the MTIM interrupt vector is properly defined.

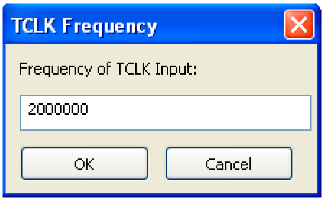
Modulo Timer Interrupt Module User Commands

The following commands are available for the MTIM.

TCLK Command

The TCLK opens the TCLK frequency dialog box shown in Figure 10.35. You may then use this box to specify the input frequency of the TCLK.

Figure 10.35 TCLK Frequency Dialog Box



Frequency of TCLK Input: 2000000

Syntax

```
>gdi TCLK
```

Example

```
>gdi TCLK
```

Show TCLK Frequency Dialog Box

TCLK <n> Command

The TCLK <n> command sets the TCLK input frequency to <n>.

Syntax

```
>gdi TCLK <n>
```

Where:

`<n>` is the input frequency of TCLK

Example

```
>gdi TCLK 200000t
```

Simulate the TCLK input frequency of 200000 Hz.

MSCAN Controller Module

The MSCAN Controller Module fully simulates the operation of the MSCAN08 Protocol Version 2.0 based device, including:

- Flag polling
- Interrupt enabled mode
- 0-8 bytes data length
- Transmission and reception of external data

The MSCAN08 peripheral is a scalable Control Area Network (CAN) 2.0 compliant device that allows microcontrollers to exchange data between themselves at high speeds. This is done through a high-speed serial link that is deterministic and reliable. CAN devices are often utilized in automobiles, where multiple microcontrollers need to be connected into a network. The CAN specification indicates that any unit on the bus can be a master at any time, which sends a message to another unit at any time, provided the bus is free to do so. All of these messages can be set up through the CAN I/O commands built into the simulator. This section goes through an example which shows how the simulator can be used to test out code that drives the CAN peripheral

Programmable Delay Block Module

In Full Chip Simulation (FCS) Mode, this module lets you simulate all the functionality of the Programmable Delay Block (PDB) module. This module's primary function is to provide a controllable delay from FTM SYNC output to the sample trigger input of PGA or ADC, or a controllable window synchronized with PWM pulses for ACMP to compare the analog signals in a defined window. PDB can alternately generate PWM pulses that are synchronized to FTM, CMPR output, and RTC.

Programmable Gain Amplifier Module

In Full Chip Simulation (FCS) Mode, this option lets you simulate all the functionality of the Programmable Gain Amplifier (PGAV1) module, including data input on all PGA channels, flag polling, and output connection to the ADC input channel. The user can utilize either the PGA Inputs display form or command-line commands to provide inputs to the PGA module. The PGAINPUTS command shows the simulated PGA input/output values. There are also three specific commands in the simulation for providing PGA inputs to simulation via a command line. They are: PGAINPLUS <x>, PGAINMINUS <x>, PGAINVDDA <x>. These commands allow the user to automate the testing/debugging procedure without relying on a modal form for entering the data.

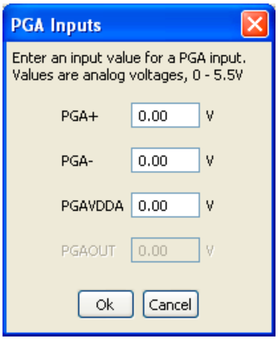
Programmable Gain Amplifier User Commands

The following commands are available for the HCS08 PGAV1 Module.

PGAINPUTS Command

The PGAINPUTS command shows the simulated PGA input/output values. The user may use this window to specify the input values of PGA+, PGA-, and PGAVDDA. The read-only PGAOUT is the value of PGA output.

Figure 10.36 PGA Inputs Window



PGA Inputs

Enter an input value for a PGA input.
Values are analog voltages, 0 - 5.5V

PGA+ 0.00 V
PGA- 0.00 V
PGAVDDA 0.00 V
PGAOUT 0.00 V

The PGAINPUTS command can be used at any point to bring up this PGA Inputs window. The window displays the input values of PGA+, PGA-, PGAVDDA and the output value of PGAOUT.

Syntax

```
>gdi PGAINPUTS
```

Examples

```
>gdi PGAINPUTS
```

Pull up the PGA Inputs window

PGAINPLUS <x> Command

The PGAINPLUS <x> command allows user to input a value for PGA+ channel.

Syntax

```
>gdi PGAINPLUS <x>
```

Where:

<x> is the input value in volts

Examples

```
>gdi PGAINPLUS 2.00
```

Simulate the input 2.00 volts for the PGA+ channel

PGAINMINUS <x> Command

The PGAINMINUS <x> command allows the user to input a value for the PGA- channel.

Syntax

```
>gdi PGAINMINUS <x>
```

Where:

<x> is the input value in volts

Examples

```
>gdi PGAINMINUS 3.50
```

Simulate the input 3.50 volts for PGA- channel

PGAINVDDA <x> Command

The PGAINVDDA <x> command allows the user to input a value for the VDDA signal that the PGA module connects to.

Syntax

```
>gdi PGAINVDDA <x>
```

Where:

<x> is the input value in volts

Examples

```
>gdi PGAINVDDA 1.50
```

Simulate the input 1.50 volts for VDDA signal

Programmable Reference Analog Comparator Module

In Full Chip Simulation (FCS) Mode, this option lets you simulate all the functionality of the Programmable Reference Analog Comparator (PRACMP) module, including data input on all PRACMP channels, flag polling, and interrupt operation. The user can use either the PRACMP Inputs display form or command-line commands to provide inputs to PRACMP module.

Programmable Reference Analog Comparator User Commands

The following commands are available for the HCS08 PRACMP Module.

PR<x>INPUT<y> Command

The PR<x>INPUT<y> command allows the user to input voltage value for the external analog input CIN<y>. For HCS08 devices that have more than one PRACMP module, <x> is the number representing the corresponding module. For HCS08 devices that only have one PRACMP module, <x>=1.

Syntax

```
>gdi PR<x>INPUT<y> <n>
```

Where:

`<x>` is the number representing the corresponding PRACMP module

`<y>` is the number representing the corresponding external analog input CIN

`<n>` is the voltage input value

Examples

```
>gdi PR1INPUT1 5.00
```

Simulate the input 5.00 volts on CIN1 of PRACMP1.

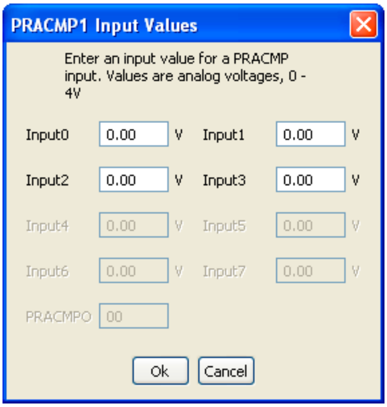
```
>gdi PR4INPUT3 2.50
```

Simulate the input 2.50 volts on CIN3 of PRACMP4.

PR<x>INPUTS Command

In FCS Mode, the PR<x>INPUTS command opens the PRACMP<x> Input Value dialog box shown in Figure 10.37. The user may then use this box to specify the external analog input voltages. For HCS08 devices that have more than one PRACMP module, <x> is the number representing the corresponding module. For HCS08 devices that only have one PRACMP module, <x>=1.

Figure 10.37 PRACMPx Input Value dialog box



PRACMP1 Input Values

Enter an input value for a PRACMP input. Values are analog voltages, 0 - 4V

Input0 0.00 V
Input1 0.00 V
Input2 0.00 V
Input3 0.00 V
Input4 0.00 V
Input5 0.00 V
Input6 0.00 V
Input7 0.00 V
PRACMPO 00

When using FCS, the PR<x>INPUTS command shows the simulated input analog voltages to any applicable PRACMPx module.

Syntax

```
>gdi PR<x>INPUTS
```

Where:

<x> is the number representing the corresponding PRACMP module

Example

```
>gdi PR1INPUTS
```

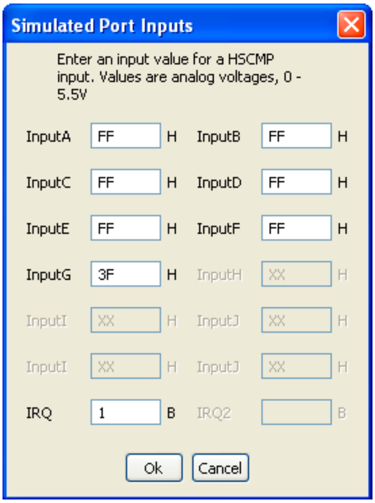
Show input analog voltages of PRACMP1.

Input/Output (I/O) Ports Module

In Full Chip Simulation (FCS) mode, this module simulates all input and output functionality of the Input/Output (I/O) Ports module. FCS mode uses a set of designated commands to simulate the input and output activity on corresponding I/O port pins. To define an input state of a specific port, write the INPUT <x> <n> command in the Command window. The <x> represents the corresponding I/O port, while the <n> stands for the input value to write to this port. At the same time, you can use the INPUTS command to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports.

NOTE Refer the Input/Output Ports User Commands and IRQ Commands for more information about the various forms of this command.

Figure 10.38 Simulated Port Inputs Dialog Box

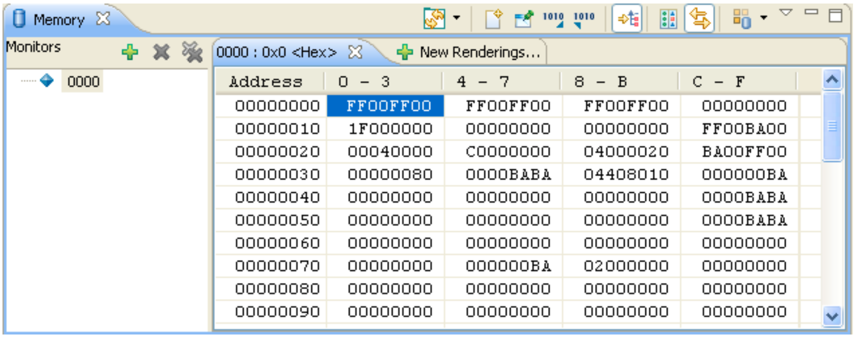


Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA	FF	H	InputB	FF	H
InputC	FF	H	InputD	FF	H
InputE	FF	H	InputF	FF	H
InputG	3F	H	InputH	XX	H
InputI	XX	H	InputJ	XX	H
InputI	XX	H	InputJ	XX	H
IRQ	1	B	IRQ2		B

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. Use the INPUTS command to reconfigure the output values on any relevant I/O port. You can observe the manipulation of I/O port pins in the Memory window.

Figure 10.39 Memory Component Window



Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

Note that if the regular I/O pins are multiplexed to be used by a different MCU Module, they might not be available for general I/O functionality.

NOTE For more information on how to properly configure I/O pins, refer to the Freescale user manual for your microprocessor.

Input/Output Ports User Commands

Use the following commands for general I/O ports manipulation.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example

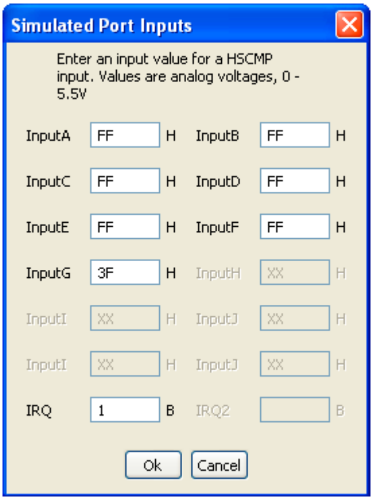
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 10.40. You may then use this box to specify the input states of port pins and IRQ.

Figure 10.40 Simulated Port Inputs Dialog Box



Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA FF H InputB FF H
InputC FF H InputD FF H
InputE FF H InputF FF H
InputG 3F H InputH XX H
InputI XX H InputJ XX H
InputI XX H InputJ XX H
IRQ 1 B IRQ2 B

When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

Serial Communications Interface Module

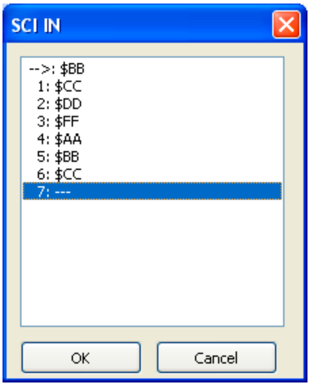
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- 8- or 9-bit length data codes
- Odd and even parity modes
- Transmission and reception of external data

FCS mode uses the buffered input/output structure to simulate SCI inputs. You can queue up to 256 data values into the input buffer. The output buffer of the SCI module can also hold 256 output values. To queue the SCI Input Data, use the SCDI <n> command in the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer.

Otherwise, if no parameter is provided, a window is displayed with the input buffer values. You can enter input values while the window is open. An arrow points to the next value to be used as input to the SCI. The data from the SCI input buffer is written to the SCI data register once the SCI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SCI IN Buffer.

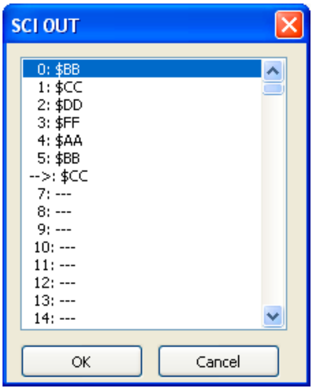
Figure 10.41 SCI IN Buffer Display



```
-->: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
  6: $CC
  7: ---
```

SCI Data Output Buffer simulation lets you gain access to the past 256 SCI data values transmitted out of the module. To bring up the SCI OUT buffer dialog box, use the SCDO command.

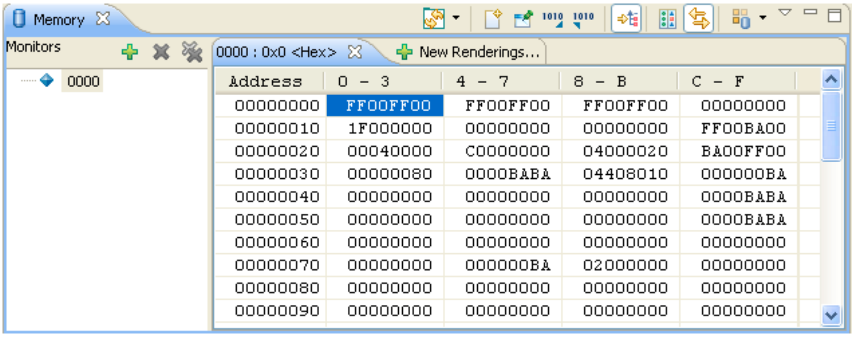
Figure 10.42 SCI OUT Buffer Display



```
  0: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
-->: $CC
  7: ---
  8: ---
  9: ---
 10: ---
 11: ---
 12: ---
 13: ---
 14: ---
```

The SCCLR command may be used at any point to flush the input and output SCI buffers. After the SCI simulated input is received, the first queued value is passed from the data buffer into the SCI data register. It can be observed in the memory window by displaying the memory location corresponding to the SCI data register.

Figure 10.43 Memory Component Window



Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

You can also observe different SCI flags in the Memory window. If the module is run in Flag Polling mode, poll the flag corresponding to the expected SCI event. If the SCI interrupts are enabled, the FCS jumps to an appropriate subroutine as long as the SCI interrupt vectors are properly defined.

NOTE For more information on how to configure the SCI module for desired operation, refer to the Freescale user manual for your microprocessor.

SCI Commands

The following serial communication interface commands are available for the HC08/ HCS08.

SCCLR Command

Use the SCCLR command to flush the input and output buffers for SCI simulation. This resets the buffers and clears out all values. Note that if the SCI is in the process of shifting a value, this command allows the SCI to finish the transfer. Refer the SCDI and SCDO commands for accessing the input and output buffers of the SCI interface.

Syntax

```
>gdi SCCLR
```

Example

```
>gdi SCCLR
```

Clear input and output buffer for SCI simulation

SCDI Command

The SCDI command lets you input data into the SCI. If a data parameter is given, the value is placed into the next slot in the SCI input buffer. If no parameter is given, a window displays the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used as input to the SCI. The maximum number of input values is 256 bytes.

Syntax

```
>gdi SCDI [<n>]
```

Where:

`<n>` The value to be entered into the next location in the input buffer

Example

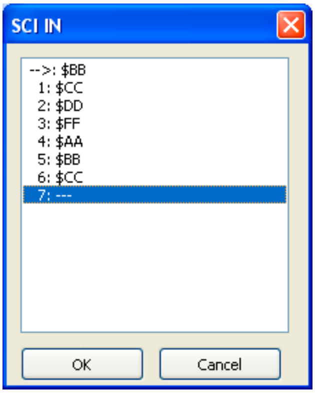
```
>gdi SCDI $55
```

Set the next input value to the SCI to $55

```
>gdi SCDI
```

Pull up the data window with all the input values.

Figure 10.44 SCI IN Buffer Display



```
-->: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
  6: $CC
  7: ---
```

SCDO Command

The `SCDO` command displays the output buffer from the SCI. A window is opened that shows all the data that the SCI has shifted out. An arrow points to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax

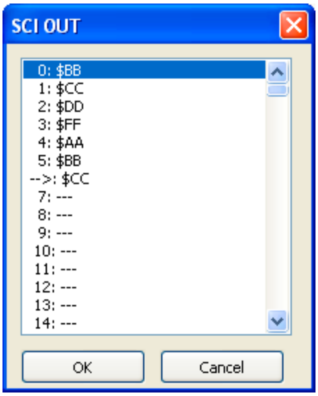
```
>gdi SCDO
```

Example

```
>gdi SCDO
```

View data from the output buffer for the SCI simulation.

Figure 10.45 SCI OUT Buffer Display



```
0: $BB
1: $CC
2: $DD
3: $FF
4: $AA
5: $BB
-->: $CC
7: ---
8: ---
9: ---
10: ---
11: ---
12: ---
13: ---
14: ---
```

Slave LIN Interface Controller (SLIC) Module

In Full Chip Simulation (FCS) mode, this option will simulate all functionality of the Slave LIN Interface Controller (SLIC) Module, including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Check sum generation and verification
- Different message lengths data modes

Full Chip Simulation (FCS) mode uses a buffered structure to simulate SLIC inputs and outputs. You can queue up to 256 data bytes into the input buffer. The output buffer of the SLIC module can also hold 256 output bytes. To queue the SLIC Input bytes use the SLCIN instruction in the command prompt. For a more detailed description of the command, please refer to the SLIC Commands section. The SLIC instruction brings up a window, which displays a list of queued input data. Different SLIC packets can be entered while the window is open. An arrow points to the byte that will be used next as input to the SLIC. Once the SLIC module is turned on and properly configured for receiving data from an external SLIC device, the data from the SLIC input buffer is written to the SLIC module identifier or data registers. After the simulation of the data transmission is complete, the arrow moves to the next value in the SLIC IN Buffer.

Serial Peripheral Interface Module

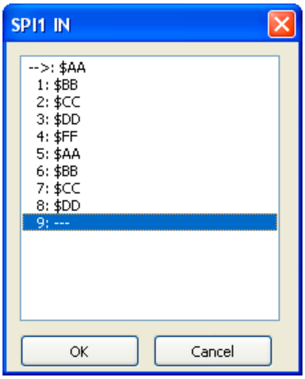
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- Master and slave modes
- Slave input clock
- Transmission and reception of external data

FCS mode uses the buffered input/output structure to simulate SPI inputs. You can queue up to 256 data values into the input buffer. The output buffer of the SPI module can also hold 256 output values. To queue the SPI Input Data, use the SPDI <n> command at the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer.

Otherwise a window is displayed with the input buffer values. You can enter input values while the window is open. An arrow points to the next input value to the SPI. The data from the SPI input buffer is written to the SPI data register once the SPI module is turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SPI IN Buffer.

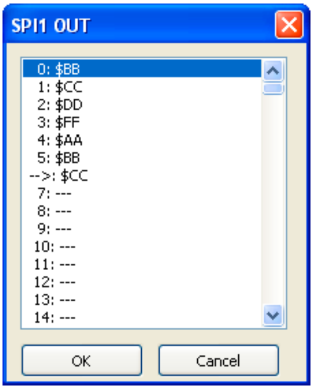
Figure 10.46 SPI IN Buffer Display



```
-->: $AA
  1: $BB
  2: $CC
  3: $DD
  4: $FF
  5: $AA
  6: $BB
  7: $CC
  8: $DD
  9: ---
```

SPI data output buffer simulation lets you gain access to the past 256 SPI data values transmitted out of the module. To bring up the SPI OUT buffer dialog box, use the SPDO command.

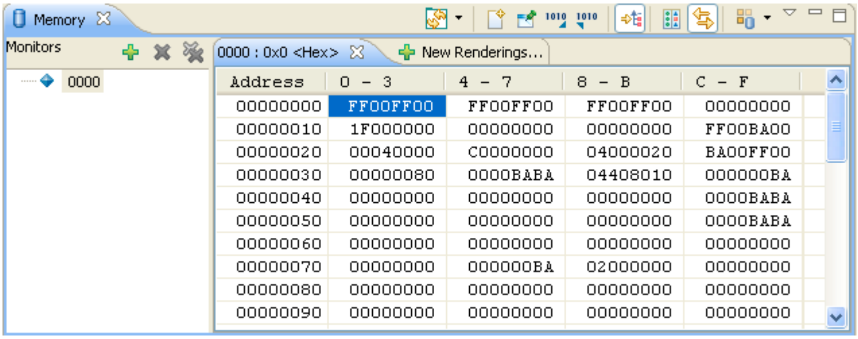
Figure 10.47 SPI OUT Buffer Display



```
0: $BB
1: $CC
2: $DD
3: $FF
4: $AA
5: $BB
-->: $CC
7: ---
8: ---
9: ---
10: ---
11: ---
12: ---
13: ---
14: ---
```

The `SPCLR` command may be used at any point to flush the input and output SPI buffers. After the SPI simulated input is received, the first queued value is passed from the data buffer into the SPI data register. It can be observed in the Memory Window by displaying the memory location corresponding to the SPI data register.

Figure 10.48 Memory Component Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

You can also observe different SPI flags in the Memory window. If the module is run in Flag Polling mode, poll the flag corresponding to the expected SPI event. If the SPI interrupts are enabled, the FCS jumps to an appropriate subroutine as long as the SPI channel interrupt vectors are properly defined.

To simulate the frequency of the SPI slave input clock, use the SPFREQ <n> command. If the SPI is configured for slave mode, this command let you enter the number of cycles <n> in the period of the input clock. If the SPFREQ command is not used, then clocking is set by the SPI control register.

NOTE For more information on how to configure the SPI module for desired operation, refer to the Freescale user manual for your microprocessor.

SPI Commands

The following serial peripheral interface commands are available for the HCS08.

SPCLR Command

Use the SPCLR command to flush the input and output buffers for SPI simulation. This resets the buffers and clears out all values. Notice that if the SPI is currently shifting a value, this command allows the SPI to finish the transfer. Refer the SPDI and SPDO commands for accessing the input and output buffers of the SPI interface.

Syntax

```
>gdi SPCLR
```

Example

```
>gdi SPCLR
```

Clear input and output buffer for SPI simulation

SPDI Command

The SPDI command lets you input data into the SPI. If a data parameter is given, the value is placed into the next slot in the SPI input buffer. If no parameter is given, a window displays the input buffer values. You can enter input values while the window is open. An arrow points to the next input value to the SPI. The maximum number of input values is 256 bytes.

Syntax

```
>gdi SPDI [<n>]
```

Where:

<n> The value to be entered into the next location in the input buffer

Example

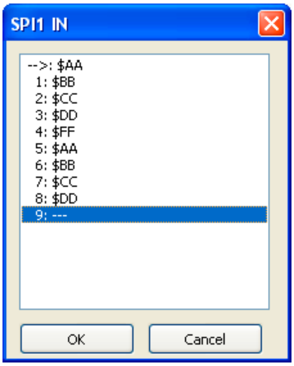
```
>gdi SPDI $55
```

Set the next input value to the SPI to $55

```
>gdi SPDI
```

Pull up the data window with all the input values.

Figure 10.49 SPI IN Buffer Display



```
-->: $AA
 1: $BB
 2: $CC
 3: $DD
 4: $FF
 5: $AA
 6: $BB
 7: $CC
 8: $DD
 9: ---
```

SPDO Command

The SPDO command displays the output buffer from the SPI. A window opens that shows all the data that the SPI has shifted out. An arrow points to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax

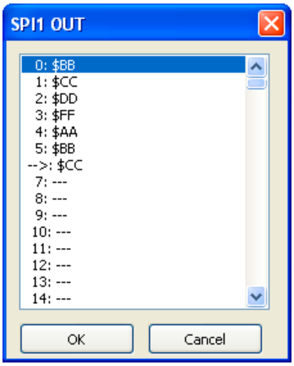
```
>gdi SPDO
```

Example

```
>gdi SPDO
```

View data from the output buffer for the SPI simulation.

Figure 10.50 SPI OUT Buffer Display



```
 0: $BB
 1: $CC
 2: $DD
 3: $FF
 4: $AA
 5: $BB
-->: $CC
 7: ---
 8: ---
 9: ---
10: ---
11: ---
12: ---
13: ---
14: ---
```

SPFREQ Command

The SPFREQ command lets you set the frequency of the SPI slave input clock. If the SPI is configured for the slave mode, this command lets you enter the number of cycles <n> per one input clock period. If no value is given, a window appears and you are prompted for a value. If this command is not used, then the clocking is assumed to be set by the SPI control register.

Syntax

```
>gdi SPFREQ [<n>]
```

Where:

`<n>` The number of cycles for the period of the input clock.

Example

```
>gdi SPFREQ 8
```

Set the period of the input slave clock to 8 cycles (total shift = 8*8 cycles per bit =

64 cycles)

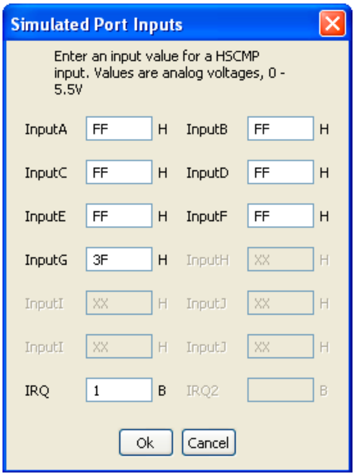
Timer Interface Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Timer Interface module, including:

- Input capture/output compare
- Pulse width modulation
- Internal or external clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation.

FCS mode uses the simulated port inputs to trigger the input capture on a given timer channel. To define an input state of the specific port, use the INPUT<x> <n> command in the Command window. The <x> represents the corresponding I/O port, while <n> stands for the input value to write to this port. At the same time, the INPUTS command can be used to display the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Refer the documentation for Timer Module Commands for more information about the various forms of this command.

Figure 10.51 Simulated Port Inputs Dialog Box



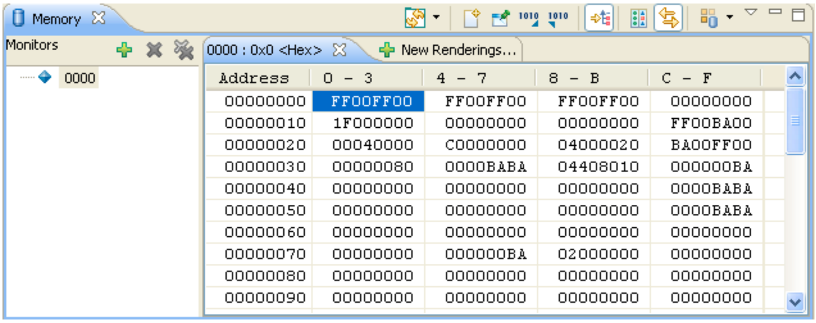
Simulated Port Inputs

Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA	FF	H	InputB	FF	H
InputC	FF	H	InputD	FF	H
InputE	FF	H	InputF	FF	H
InputG	3F	H	InputH	XX	H
InputI	XX	H	InputJ	XX	H
InputI	XX	H	InputJ	XX	H
IRQ	1	B	IRQ2		B

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. To trigger the event, first set the port inputs high or low and then invert them to an opposite value, depending on whether the input capture is set for rising/falling edge. Once the Input Capture event takes place you can observe the CHxF in the Channel Status and Control register in the Memory window.

Figure 10.52 Memory Component Window



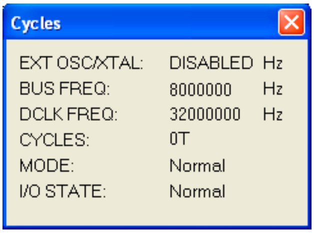
Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

If the Timer module is configured for an Output Compare event, then once the event takes place you can observe the same CHxF Flag via the Memory window. If the timer channel interrupt is enabled, the FCS jumps to an appropriate subroutine as long as the Timer channel interrupt vector is properly defined. To observe the Timer Overflow Flag (TOF) flag being set as a result of the corresponding CPU event, situate your Memory window on the memory location of the Timer Status and Control register.

To observe the Pulse Width Modulation (PWM) operation, properly configure the Timer to operate in the Modulo up count mode, select the toggle-on-overflow or clear/set output on compare events to create the desired duty cycle wave. Once a PWM event takes place, you can observe pin toggle/clear/set behavior corresponding to the Timer configuration in the Memory window displaying the I/O port associated with a given timer channel.

To observe the accuracy of the Timer module operation, you can observe the number of CPU cycles that it takes for the event to occur. The cycle counter is only incremented as the you step through the code. To determine the exact amount of cycles over which the event occurs, one can either observe the cycle display in the Register window or use the built in simulation commands. To display the current number of cycles in the Command window, use the CYCLES command. To change the number of cycles in the cycle counter, use CYCLES <n>, where <n> is the new cycle value. If the event has a pre-calculated number of cycles, use CYCLE 00 to reset the number of cycles and GOTOCYCLE <n> to run through the code until you reach the expected event.

Figure 10.53 Register Window With Cycles Display



EXT OSC/XTAL:	DISABLED	Hz
BUS FREQ:	8000000	Hz
DCLK FREQ:	32000000	Hz
CYCLES:	0T	
MODE:	Normal	
I/O STATE:	Normal	

Timer Module Commands

The following timer module commands are available for use with the HC08/HCS08 processors.

CYCLES Command

The CYCLES command changes the value of the cycles counter. The cycles counter counts the number of the processor cycles that have passed during execution. The Cycles window shows the cycle counter. The cycle count can be useful for timing procedures.

Syntax

```
>gdi CYCLES <n>
```

Where:

`<n>` Integer value for the cycles counter

Examples

```
>gdi CYCLES 0
```

Reset cycles counter

```
>gdi CYCLES 1000
```

Set cycle counter to 1000.

GOTOCYCLE Command

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the Stop button on the toolbar is pressed, until it reaches a break point, or until an error occurs.

Syntax

```
>gdi GOTOCYCLE <n>
```

Where:

<n> Cycle-counter value at which the execution stops

Example

```
>gdi GOTOCYCLE 100
```

Execute the program until the cycle counter equals 100.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example

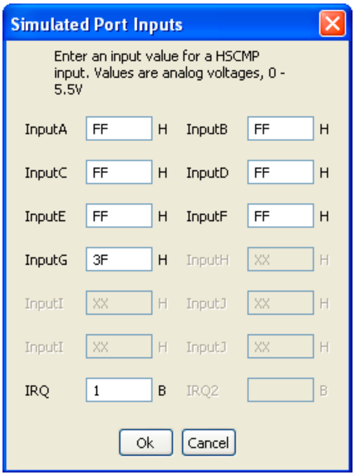
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 10.54. You may then use this box to specify the input states of port pins and IRQ.

Figure 10.54 Simulated Port Inputs Dialog Box



Simulated Port Inputs

Enter an input value for a HSCMP input. Values are analog voltages, 0 - 5.5V

InputA FF H
InputB FF H
InputC FF H
InputD FF H
InputE FF H
InputF FF H
InputG 3F H
InputH XX H
InputI XX H
InputJ XX H
InputI XX H
InputJ XX H
IRQ 1 B
IRQ2 B

When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

Time Of Day Module Option

In Full Chip Simulation (FCS) mode, this module lets you simulate all the functionality of the Time Of Day (TOD) module. The module includes an 8-bit counter, a 6-bit match register, several binary-based and decimal-based prescaler dividers, three clock source options, and one interrupt that can be used for quarter second, one second and match conditions. A 4 Hz signal is used as the reference clock for the TOD counter, where each tick of the TOD counter is 0.25 seconds. This module can be used for time-of-day, calendar, or any task scheduling functions. It can also serve as a cyclic wake up from low-power modes without the need for external components.

Universal Serial Bus (USB) Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Universal Serial Bus (USB) module including USB flags and interrupts, seven USB endpoints, USB RAM and USB reset options. While all control transactions occur through bi-directional endpoint 0, the other endpoints can be set up for data transfer in the input or output direction.

Some of the microcontrollers in the HCS08 family contain USB compliant peripheral devices. These can be low-speed or high-speed USB slave devices. This means that all USB transfers are initiated by a host (i.e. a personal computer) and that the microcontroller needs to be set up to respond with the appropriate acknowledgement messages. According to the USB specification, there are a series of messages that go back and forth between the host and the device in order to set up and describe the channel for data transfer. All of these messages can be set up through the USB I/O commands built into the simulator. This section goes through an example of this, showing how the simulator can be used to test out code for driving the USB peripheral.

Voltage Reference Module

In Full Chip Simulation (FCS) mode, this module lets you simulate all the functionality of the Voltage Reference (VREF) module. The module is a bandgap buffer system intended to supply an accurate voltage output that is trimmable by an 8-bit register in 0.5 mV steps. It can be used internally for the analog peripherals of an ADC channel or for an ACMP input. VREF has three operating modes that provide different levels of load regulation and power consumption.

P&E HCS08 Multilink\Cyclone Pro\OSBDM

This section describes the HCS08 P&E Connection options. The HCS08 P&E Connection setting permits a connection to HCS08 Freescale devices via P&E Multilink/Cyclone PRO/OSBDM hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

Connection Options

This topic describes all P&E HCS08 Multilink/Cyclone Pro/OSBDM connection options, which are common to all P&E USB BDM Multilink or USB Multilink Universal, P&E Cyclone PRO Serial, P&E Cyclone PRO USB, P&E Cyclone PRO Ethernet, and P&E Open Source BDM connections.

The connection options include:

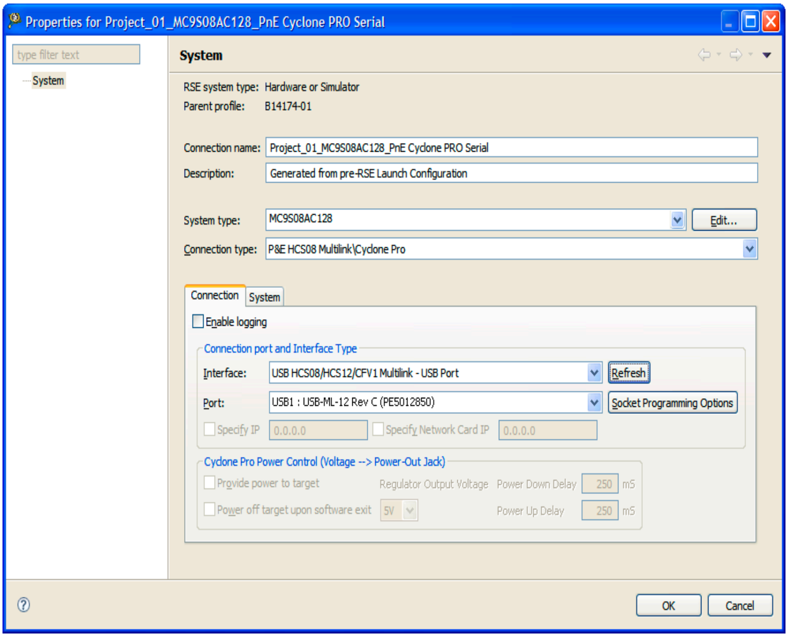
- Changing P&E Connections Settings

- Connection Assistant
- Launch Configuration Settings
- Active Mode Menu Options
- Advanced Programming/Debug Options
- View Register Files Option
- Socket Programming Options Button
- P&E HCS08 Multilink\Cyclone PRO/OSBDM Connection-Specific Options

Changing P&E Connections Settings

All connection settings for P&E hardware interfaces are configured using the **Remote System** group in the **Main** tab of the **Debugger Configuration** dialog box.

Figure 10.55 P&E Multilink/Cyclone Pro/OSBDM



Properties for Project_01_MC9S08AC128_PnE Cyclone PRO Serial

System

RSE system type: Hardware or Simulator
Parent profile: B14174-01

Connection name: Project_01_MC9S08AC128_PnE Cyclone PRO Serial
Description: Generated from pre-RSE Launch Configuration

System type: MC9S08AC128
Connection type: P&E HCS08 Multilink\Cyclone Pro

☐ Enable logging

Connection port and Interface Type
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port
Port: USB1 : USB-ML-12 Rev C (PE5012850)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☐ Provide power to target
☐ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

Table 10.1 describes the options for the **P&E Multilink/Cyclone PRO/OSBDM** connection.

Table 10.1 Connection Parameter Options for P&E HCS08 Multilink/Cyclone Pro/OSBDM

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB HCS08\HCS12\CFV1 Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone PRO - Serial Port • Cyclone PRO - USB Port • Cyclone PRO - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project. • OSBDM
Refresh	Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE:** If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help.
Socket Programming Options	The Socket Programming Options button brings up a dialog that provides you with a graphical representation of the signals that must be connected from the BDM header to the pins of the microprocessor, in order to use Freescale socket adapters.

Table 10.1 Connection Parameter Options for P&E HCS08 Multilink/Cyclone Pro/OSBDM

Option	Description
(Cyclone Ethernet only) Specify IP	Use this option to specify the IP address of a Cyclone outside of the local network. Click on the checkbox to enable the textbox. This will also disable the port dropdown box. Currently supports IPv4 only.
(Cyclone Ethernet only) Specify Network Card IP	Use this option to specify the local network card IP address if there are multiple cards on your computer. Click on the checkbox to enable the textbox. Currently supports IPv4 only.
(Cyclone Pro only) Provide power to target	This option determines whether the Cyclone Pro (circuitry) provides power to the target hardware via the probe. Check this option to have the Cyclone Pro (circuitry) supply power to the hardware target Uncheck this option to not provide power.
(Cyclone Pro only) Power off target upon software exit	This option determines whether the Cyclone PRO hardware interface provides power to the target hardware via the VDD of the BDM cable. Check this option to turn off the power when the program terminates. Uncheck this option to leave the hardware target powered continuously.
(Cyclone PRO only) Regulator Output Voltage	This option adjusts the output voltage that powers the hardware target. Select a voltage value from this option’s list box.

Table 10.1 Connection Parameter Options for P&E HCS08 Multilink/Cyclone Pro/OSBDM

Option	Description
(Cyclone PRO only) Power down delay	This option specifies amount of time for which the target will be turned off during a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option’s text box.
(Cyclone PRO only) Power up delay	This option specifies amount of time for which the target will remain powered prior to a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option’s text box.

WARNING! An improper voltage setting can damage the board.

To change P&E Connections settings, perform these steps.

1. In the **CodeWarrior Projects** view, select the project for which you want to change the P&E Connections settings.

NOTE It is assumed that you have created a project and built it.

2. Select **Run > Debug Configurations** from the main menu bar of the IDE.

   The **Debug Configurations** dialog box appears.
3. Expand the **CodeWarrior Download** tree control in the left pane and select the launch configuration you want to debug.
4. Click the **Main** tab.

   The **Main** page appears in the area beneath the tabs.
5. Select a system within **Remote System** which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the Freescale Eclipse Extensions Guide. Once a remote system is selected, click the **Edit** button. The **Launch Configuration Settings Panel** will appear.
6. Ensure that the **System Type** is the correct MCU you want to debug. Use the drop-down box or the **Edit** button to change this option.
7. In the **Connection Type** drop-down box, select P&E HCS08 Multilink\Cyclone Pro. The P&E connections settings will appear below.

8. Click **Refresh** to scan valid interface and port.

   Valid interfaces and ports appear in the **Interface** and **Port** drop-down lists in the **Connection** port and **Interface Type** group.

9. Select a supported interface from the **Interface** drop-down list.
10. Select a supported port from the **Port** drop-down list.

NOTE The port displayed may vary depending on the interface. For example, if you select interface as Cyclone PRO - Serial Port, the available port option is COM1 : Serial Port 1.

11. Specify settings in the **Cyclone Pro Power Control** (**Voltage --> Power -Out Jack**) group.

NOTE This group will be enabled for Cyclone PRO interface only.

- Check the **Provide power to target** checkbox to have the Cyclone PRO (circuitry) provide power to the target else clear the checkbox if you do not want to provide power to the target.
- Check the **Power off target upon software exit** checkbox to turn off the power when the program terminate else clear the checkbox to leave the hardware target powered continuously.
- Select a voltage value from the **Regulator Output Voltage** drop-down list. This adjusts the output voltage that powers the hardware target.

WARNING! An improper voltage setting can damage the board.

- Enter the delay interval (in milliseconds) in the **Power Down Delay** text box. This option specifies the time interval to wait before shutting off the power to the hardware target. Cyclone Pro powers down the device once the debug session is over, or while executing a power cycling sequence after beginning a new debug session.
- Enter the delay interval (in milliseconds) in the **Power Up Delay** text box. This option specifies the time interval to wait before turning on the power to the hardware target. If the Cyclone Pro power to target feature is enabled, Cyclone Pro will power up the device while executing a power cycling sequence at the beginning of every debug session.

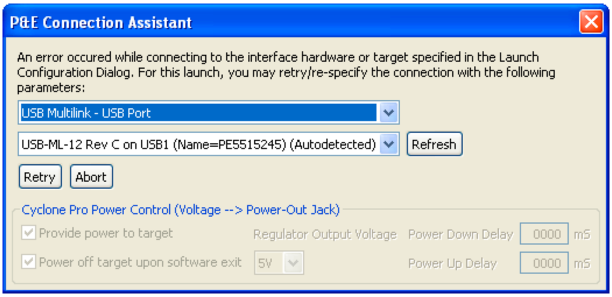
12. Click **OK** to save changes to the P&E Connections settings. The Launch Configuration Settings Panel dialog box will close.
13. Click **Close** button to close the Debug Configuration dialog box.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E HCS08 Multilink/Cyclone PRO/OSBDM** as your debugger connection:

1. Select the **P&E** device that you are using from the first drop-down list and click **Refresh**. Refer Figure 10.56.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings (Cyclone Pro only). Refer to Table 10.1 for more details of each setting.
4. Click the **Retry** button.

Figure 10.56 HCS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

Launch Configuration Settings

To set the launch configurations for the debugger:

1. Find the debugger icon and click the drop-down arrow to bring up the debugger menu. Refer Figure 10.57.
2. Select **Debug Configurations**.
3. In the left column, select the project download type you would like to set the launch configurations. Refer Figure 10.58
4. In the right column, click the **Debugger** tab.
5. Set your configurations and click the Debug button to start the debugger.

Figure 10.57 Debugger Drop-Down List

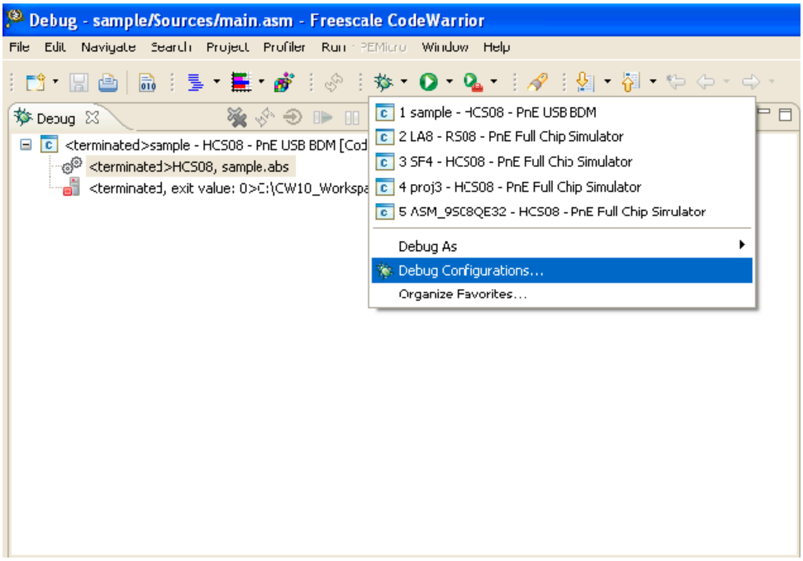
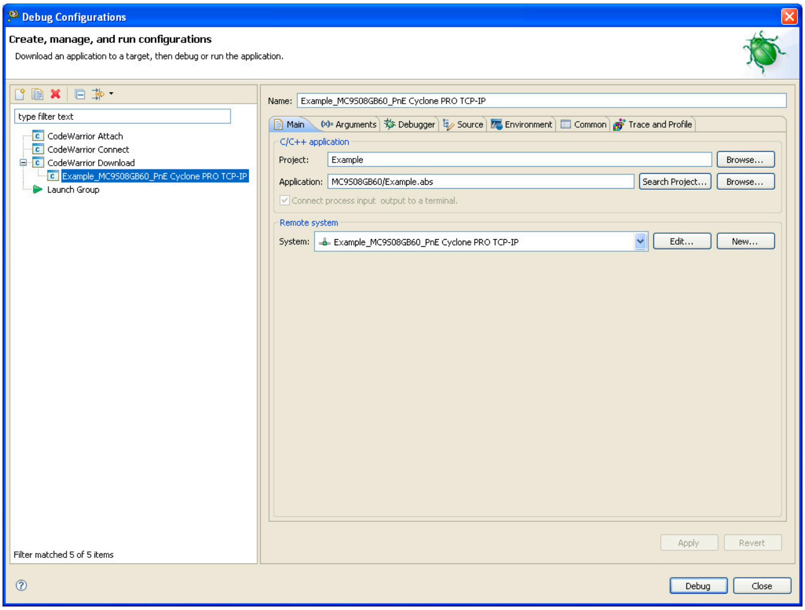

Debug - sample/Sources/main.asm - Freescale CodeWarrior
File Edit Navigate Search Project Profiler Run PEMicro Window Help
Debug
<terminated>sample - HCS08 - PnE USB BDM [Cod
<terminated>HCS08, sample.abs
<terminated, exit value: 0>C:\CW10_Workspa
1 sample - HCS08 - PnE USB BDM
2 LA8 - RS08 - PnE Full Chip Simulator
3 SF4 - HCS08 - PnE Full Chip Simulator
4 proj3 - HCS08 - PnE Full Chip Simulator
5 ASM_9SC8QE32 - HCS08 - PnE Full Chip Simulator
Debug As
Debug Configurations...
Organize Favorites...

Figure 10.58 Debug Configuration Dialog Box



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: Example_MC9S08GB60_PnE Cyclone PRO TCP-IP

Project: Example

Application: MC9S08GB60/Example.abs

☑ Connect process input output to a terminal.

System: Example_MC9S08GB60_PnE Cyclone PRO TCP-IP

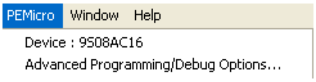
Filter matched 5 of 5 items

type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
Example_MC9S08GB60_PnE Cyclone PRO TCP-IP
Launch Group
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
C/C++ application
Browse...
Search Project...
Browse...
Remote system
Edit...
New...
Apply
Revert
Debug
Close

Active Mode Menu Options

When the microprocessor is connected, the active mode menu shows the name of the microprocessor and gives you the access to the Advanced Programming/Debug Options. When the microprocessor is not connected, the menu is not available.

Figure 10.59 Additional Connection Menu Options

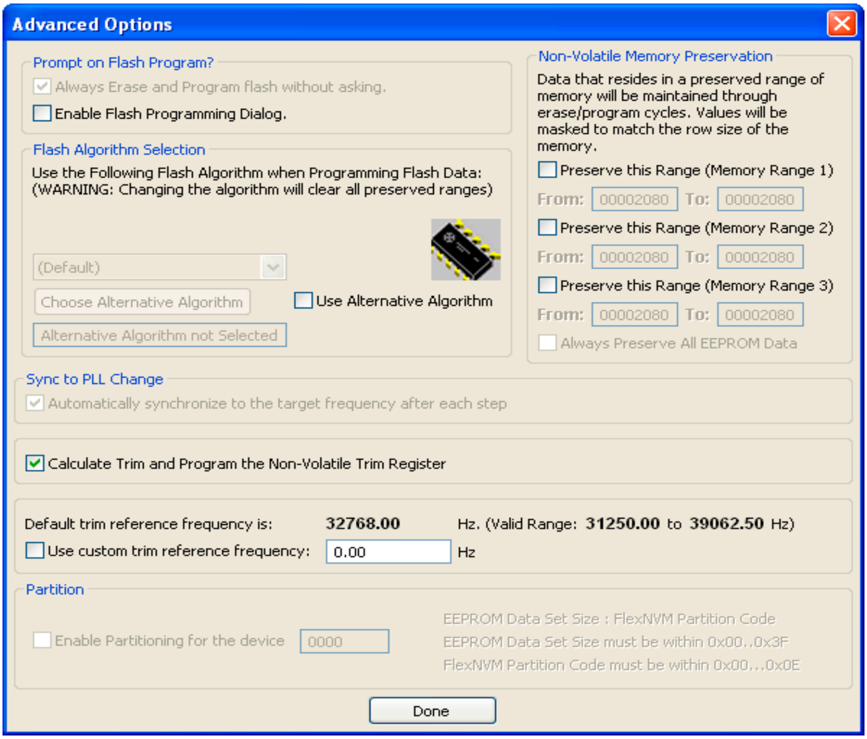


PEMicro
Window
Help
Device : 9S08AC16
Advanced Programming/Debug Options...

Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the flash programming procedure.

Figure 10.60 Advanced Options Dialog Box


Advanced Options
Prompt on Flash Program?
Flash Algorithm Selection
Non-Volatile Memory Preservation
Sync to PLL Change
Partition
Choose Alternative Algorithm
Done

☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

☐ Use Alternative Algorithm

Alternative Algorithm not Selected

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 2)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 3)
From: 00002080 To: 00002080
☐ Always Preserve All EEPROM Data

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: **32768.00** Hz. (Valid Range: **31250.00** to **39062.50** Hz)
☐ Use custom trim reference frequency: 0.00 Hz

☐ Enable Partitioning for the device 0000

EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Enable Flash Programming Dialog

Setting the Enable Flash Programming dialog box lets you view the steps taken by the Flash Programmer.

Trim Options

The Calculate Trim and Program the Non-Volatile Trim Register checkbox enables automatic calculation and programming of the trim value to a designated Non-Volatile memory location.

Non-Volatile Memory Preservation

You have the option of preserving up to three independent ranges of non-volatile memory (on devices with EEPROM, the entire EEPROM array may optionally be preserved as well). Ranges that are designated as “preserved” are read before an erase, and re-programmed immediately afterwards, thereby preserving the data in these ranges. Any attempt to program data into a preserved range is ignored. When entering an address into the preserved range field (hexadecimal input is expected), the values are masked according to the row size of the device. This ensures that the reprogramming of preserved data does not cause any conditions that disturb programming.

Sync to PLL Change Checkbox

The debugger requires that Sync to PLL Change be selected to synchronize the software/hardware connection with the microprocessor during the Flash erasing/programming procedure. This option is always enabled for M68HCS08 devices.

Trim Control

The Use custom trim reference frequency option lets you select a custom trim value for the target device (valid only for devices with an Internal Clock). The allowable trim value is limited only by the device itself; you can input any value within the valid internal clock frequency range. Note that the valid internal clock frequency range and the default trim value for the currently selected device/algorithm are displayed as well. For more information about the specific functionality of the internal clock source, see the Freescale Data Sheet for your specific device.

Alternative Algorithm Functionality

Once you create a project for a specific HCS08/RS08/CFV1 microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in the `<CW_Install>/MCU/bin/plugins/support/HC08/gdi/P&E directory`

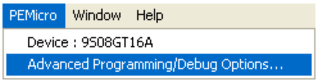
However, you can override the default algorithm via the Alternative Algorithm function, located in the Advanced Programming/Debug Options menu. This feature can be used to select a custom programming algorithm, or select another one of P&E's many programming algorithms for use with a specific project.

TIP Selecting a wrong programming algorithm may damage you device, lead to under/over programming situations, or simply not program portions of the project file. Therefore it is recommended to use the default algorithm unless there is a compelling reason to do otherwise.

Use these steps to override the default algorithm:

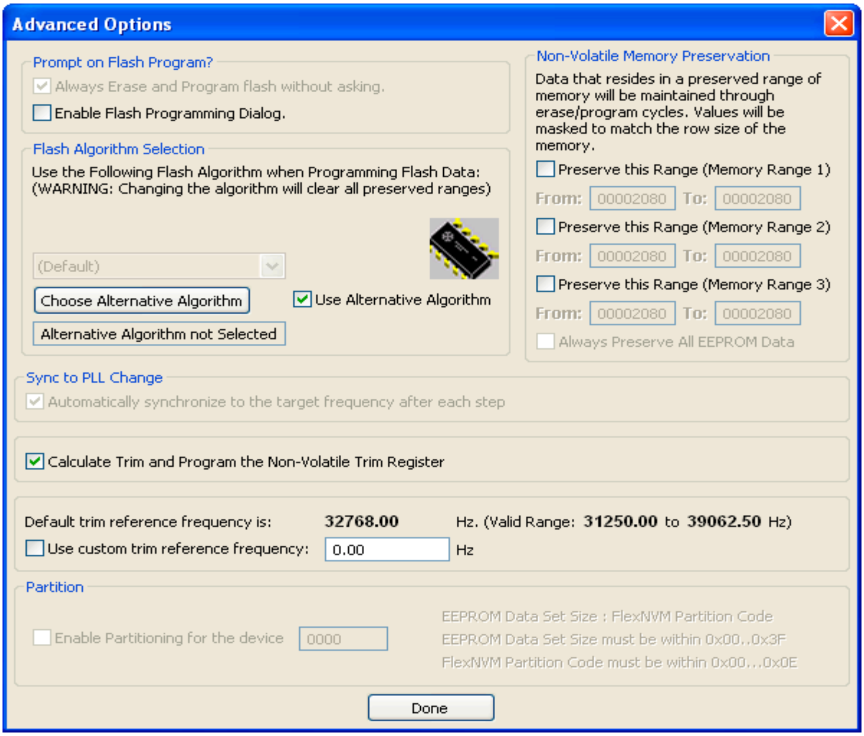
1. Select the Advanced Programming/Debug Options selection from the PEMicro menu.

Figure 10.61 Advanced Programming/Debug Options Menu Selection



2. Check the **Use Alternative Algorithm** checkbox.

Figure 10.62 Advanced Options - Alternative Algorithm Checkbox



Advanced Options

Prompt on Flash Program?
☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)
(Default)
Choose Alternative Algorithm
☑ Use Alternative Algorithm
Alternative Algorithm not Selected

Non-Volatile Memory Preservation
Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.
☐ Preserve this Range (Memory Range 1)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 2)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 3)
From: 00002080 To: 00002080
☐ Always Preserve All EEPROM Data

Sync to PLL Change
☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 32768.00 Hz. (Valid Range: 31250.00 to 39062.50 Hz)
☐ Use custom trim reference frequency: 0.00 Hz

Partition
☐ Enable Partitioning for the device 0000
EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

3. Click the **Choose Alternative Algorithm** button, which lets you browse for an alternative algorithm.

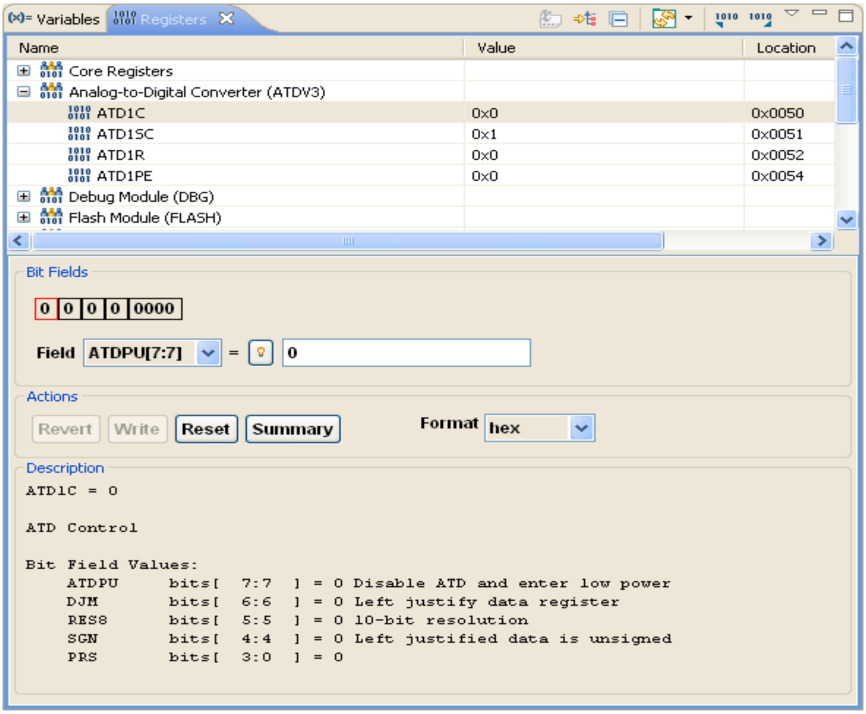
4. Once you select the alternative algorithm, the name of the algorithm along with its full path appears in the text field below the Choose Alternative Algorithm button.

At this point, the current project performs all future Flash programming operations using the alternative algorithm. You may revert to the default algorithm at any time by clearing the Use Alternative Algorithm checkbox.

View Register Files Option

The **Register Files** tab in the debugger gives you the option of viewing and editing the register files. If register files are available for the device that you have chosen, the Registers tab in the debugger (see Figure 10.63) is populated.

Figure 10.63 Debug Register File Tab



Name	Value	Location
⊞ Core Registers		
⊟ Analog-to-Digital Converter (ATDV3)		
ATD1C	0x0	0x0050
ATD1SC	0x1	0x0051
ATD1R	0x0	0x0052
ATD1PE	0x0	0x0054
⊞ Debug Module (DBG)		
⊞ Flash Module (FLASH)		

0 | 0 | 0 | 0 | 0000

Field ATDPU[7:7] = 0

Format hex

```
ATD1C = 0

ATD Control

Bit Field Values:
    ATDPU      bits[  7:7   ] = 0 Disable ATD and enter low power
    DJM        bits[  6:6   ] = 0 Left justify data register
    RES8       bits[  5:5   ] = 0 10-bit resolution
    SGN        bits[  4:4   ] = 0 Left justified data is unsigned
    PRS        bits[  3:0   ] = 0
```

To view the Register Files of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window
2. Select the **Registers** tab on the right side of the debugging window, or select the **Window > Show View > Registers** to open the **Register** window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the Registers tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The Actions box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The Description box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

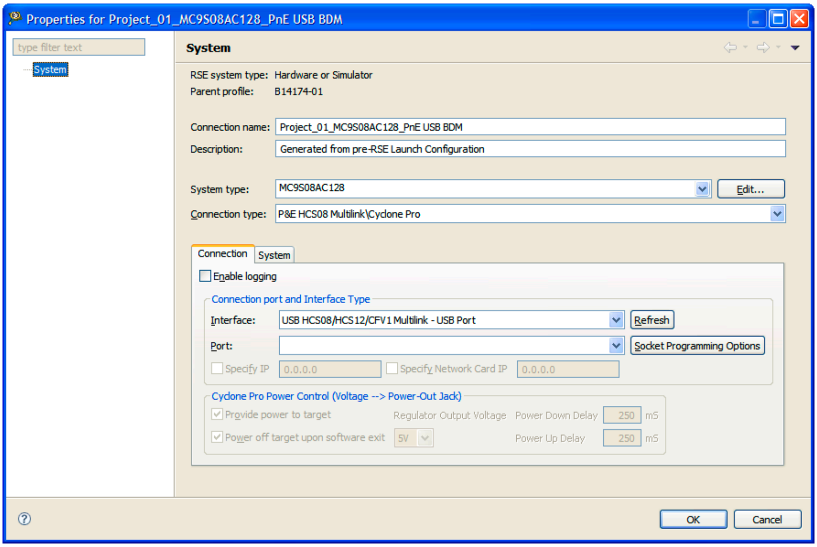
You can modify each bit by selecting it in the drop-down list under the Bit field, or by clicking the bit on the Bit field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

Socket Programming Options Button

The Programming Adapter Connections dialog assistant is designed to facilitate the use of an extensive set of Freescale programming socket adapters. This dialog can be used to get a graphical representation of the signals that must be connected from the BDM header to the pins of the microprocessor. Making these connections lets you establish communication with a given device via a hardware debug interface.

The Socket Programming Options button in the BDM Launch Configuration dialog box (see Figure 10.64) takes you to the Programming Adapter Connections dialog box (see Figure 10.65), where you can look up pin connection settings for the selected package type of the target processor. Only available package types for each target processor are listed in the Package drop-down list. Once you have selected a package type, the Adapter Information section provides the part number of the adapter board, the socket number where the processor should be placed, and a pair of header numbers that indicate which connections should be made between them. Immediately below the Adapter Information section you will find a pin layout that displays the required connections between the aforementioned pair of headers.

Figure 10.64 HCS08 BDM Launch Configuration Dialog Box



Properties for Project_01_MC9S08AC128_PnE USB BDM

type filter text

System

System

RSE system type: Hardware or Simulator
Parent profile: B14174-01

Connection name: Project_01_MC9S08AC128_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MC9S08AC128 Edit...
Connection type: P&E HCS08 Multilink\Cyclone Pro

Connection | System

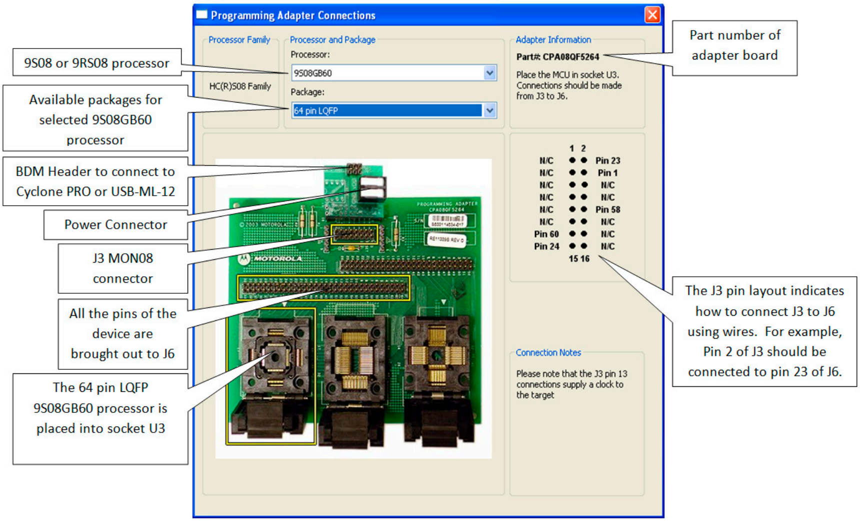
☐ Enable logging

Connection port and Interface Type
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port Refresh
Port: Socket Programming Options
☐ Specify IP 0.0.0.0 ☐ Specify Network Card IP 0.0.0.0

Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☑ Provide power to target Regulator Output Voltage Power Down Delay 250 mS
☑ Power off target upon software exit 5V Power Up Delay 250 mS

OK Cancel

Figure 10.65 Programming Adapter Connections Dialog Box



Programming Adapter Connections

Processor Family: HC(R)S08 Family

Processor and Package
Processor: 9S08GB60
Package: 64 pin LQFP

Adapter Information
Part#: CPA08QF5264
Place the MCU in socket U3. Connections should be made from J3 to J6.

	1	2	
N/C	●	●	Pin 23
N/C	●	●	Pin 1
N/C	●	●	N/C
N/C	●	●	N/C
N/C	●	●	Pin 58
N/C	●	●	N/C
Pin 60	●	●	N/C
Pin 24	●	●	N/C
	15	16	

Connection Notes
Please note that the J3 pin 13 connections supply a clock to the target

9S08 or 9RS08 processor

Available packages for selected 9S08GB60 processor

BDM Header to connect to Cyclone PRO or USB-ML-12

Power Connector

J3 MON08 connector

All the pins of the device are brought out to J6

The 64 pin LQFP 9S08GB60 processor is placed into socket U3

Part number of adapter board

The J3 pin layout indicates how to connect J3 to J6 using wires. For example, Pin 2 of J3 should be connected to pin 23 of J6.

P&E HCS08 Multilink\Cyclone PRO/OSBDM Connection-Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB BDM Multilink or USB Multilink Universal
- P&E Cyclone PRO Serial
- P&E Cyclone PRO USB
- P&E Cyclone PRO Ethernet
- P&E Open Source BDM

P&E USB BDM Multilink or USB Multilink Universal

The P&E USB BDM Multilink or USB Multilink Universal Connection setting permits a connection to USB BDM Multilink devices. P&E USB BDM Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

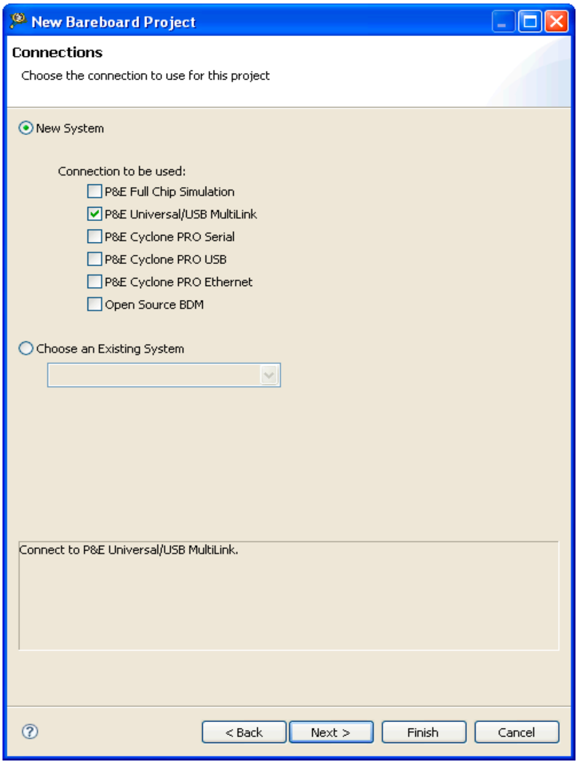
To select **P&E USB BDM Multilink** or **USB Multilink Universal** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar. The **Device/ Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next. The **Devices** page appears.
4. Expand the **HCS08** tree control and select the derivative or board you would like to use. For example, select **HCS08 > HCS08D Family > 9S08DE32**.
5. Click **Next**. The **Connections** page appears.
6. Check the **P&E Universal/USB Multilink** checkbox. Refer Figure 10.66.

Figure 10.66 HCS08 P&E USB BDM Multilink Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☑ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

7. Click **Finish**.

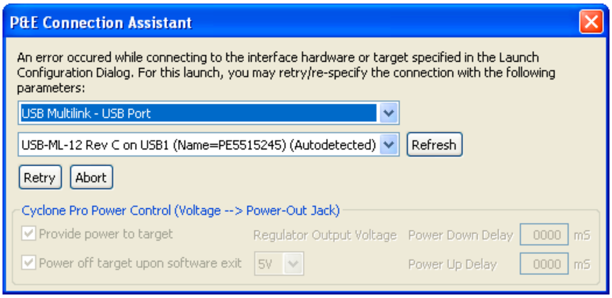
The wizard creates a project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E USB BDM Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer Figure 10.67.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 10.67 HCS08 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO Serial

The P&E Cyclone PRO Serial Connection setting permits a connection to Cyclone PRO Serial devices. P&E Cyclone PRO Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select P&E Cyclone PRO Serial as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**. The **Devices** page appears.

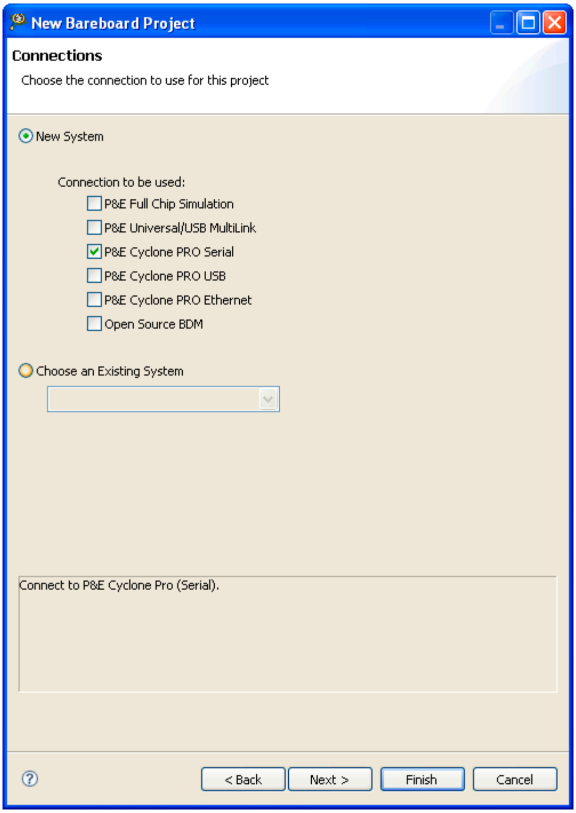
4. Expand the **HCS08** tree control and select the derivative or board you would like to use. For example, select **HCS08 > HCS08D Family > 9S08DE32**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO Serial** checkbox. Refer Figure 10.68.

Figure 10.68 HCS08 P&E Cyclone PRO Serial Selected


New Bareboard Project
Connections
Choose the connection to use for this project
● New System
Connection to be used:
☐ P&E Full Chip Simulation
☐ P&E Universal/USB MultiLink
☑ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Open Source BDM
○ Choose an Existing System
< Back
Next >
Finish
Cancel

Connect to P&E Cyclone Pro (Serial).

7. Click **Finish.**

The wizard creates a project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

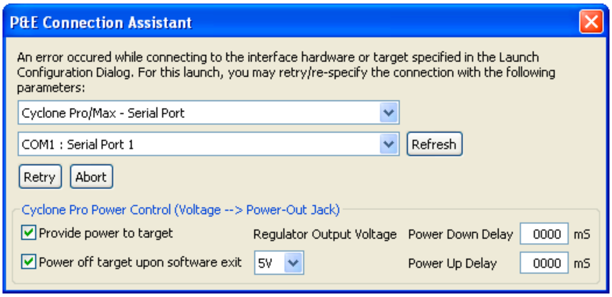
Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down list and click **Refresh**. Refer Figure 10.69.
2. Using the second drop-down list, select the port on which the interface is connected.

3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 10.1 for more details of each setting.
4. Click the **Retry** button.

Figure 10.69 HCS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port
COM1 : Serial Port 1

☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage: 5V
Power Down Delay: 0000 mS
Power Up Delay: 0000 mS

P&E Cyclone PRO USB

The P&E Cyclone PRO USB Connection setting permits a connection to Cyclone PRO USB devices. P&E Cyclone PRO USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone PRO USB** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

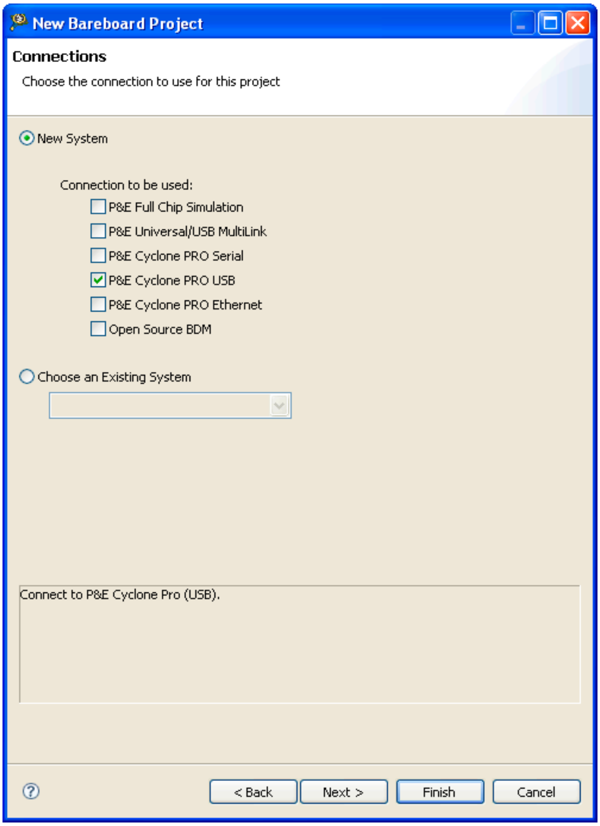
   The **Devices** page appears.
4. Expand the **HCS08** tree control and select the derivative or board you would like to use. For example, select **HCS08 > HCS08D Family > 9S08DE32**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO USB** checkbox. Refer Figure 10.70.

Figure 10.70 HCS08 P&E Cyclone PRO USB Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Full Chip Simulation
☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☑ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (USB).

7. Click **Finish**.

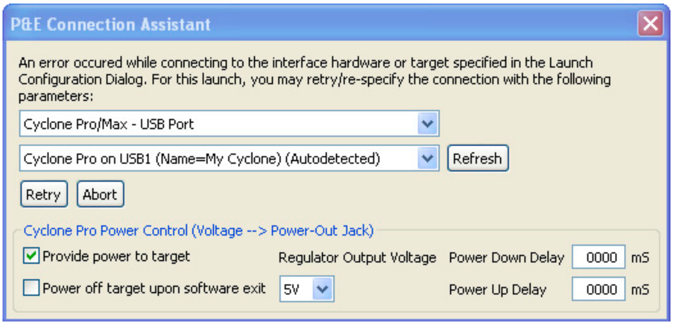
The wizard creates a project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO USB** as your debugger connection:

1. Select **Cyclone Pro/Max – USB Port** from the first drop-down list and click **Refresh**. Refer Figure 10.71.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 10.1 for more details of each setting.
4. Click the **Retry** button.

Figure 10.71 HCS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Pro on USB1 (Name=My Cyclone) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☐ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO Ethernet

The P&E Cyclone PRO Ethernet Connection setting permits a connection to Cyclone PRO Ethernet devices. P&E Cyclone PRO Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone PRO Ethernet** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

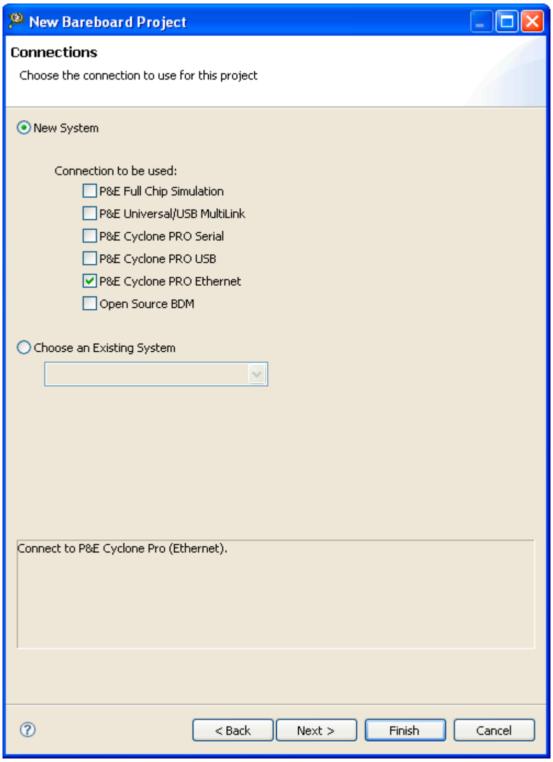
3. Click **Next**. The **Devices** page appears.
4. Expand the **HCS08 t**ree control and select the derivative or board you would like to use. For example, select **HCS08 > HCS08D Family > 9S08DE32**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO Ethernet** checkbox. Refer Figure 10.72.

Figure 10.72 HCS08 P&E Cyclone PRO Ethernet Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Full Chip Simulation
☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☑ P&E Cyclone PRO Ethernet
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (Ethernet).

7. Click **Finish.**

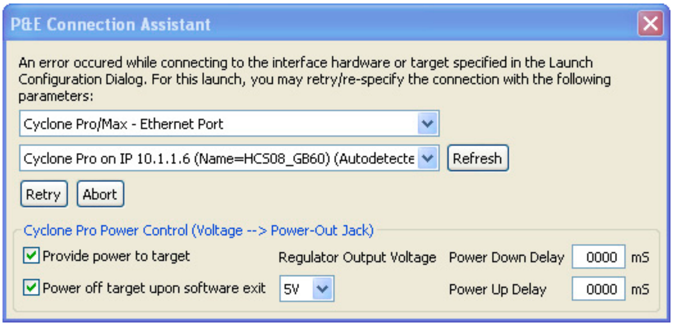
The wizard creates a project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down list and click **Refresh**. Refer Figure 10.73.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 10.1 for more details of each setting.
4. Click the **Retry** button.

Figure 10.73 HCS08 Connection Assistant Interface Selected


P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Pro on IP 10.1.1.6 (Name=HCS08_GB60) (Autodetecte

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Open Source BDM

Freescale supplies certain development boards with an integrated debug circuit based on Open Source BDM. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Pro or USB Multilink. The development board also derives its power from the USB Bus.

The Open Source BDM circuit design (OSBDM-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source BDM" connection. P&E has integrated the Open Source BDM support into the same connection that supports both the USB Multilink and the Cyclone Pro. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSBDM (albeit at a lower data rate).

The P&E Open Source BDM Connection setting permits a connection to Open Source BDM devices. Open Source BDM mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **Open Source BDM** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

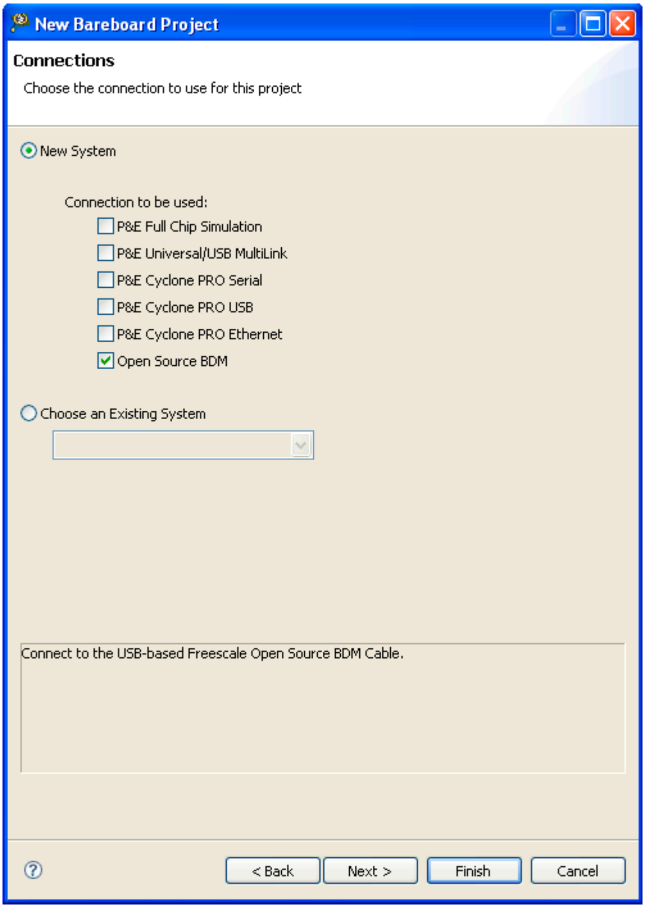
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next. The **Devices** page appears.
4. Expand the **HCS08** tree control and select the derivative or board you would like to use. For example, select **HCS08 > HCS08D Family > 9S08DE32**.
5. Click **Next**.

   The **Connections** page appears.
6. Check the **Open Source BDM** checkbox. Refer to Figure 10.74.

Figure 10.74 HCS08 Open Source BDM Selected


New Bareboard Project
Connections
Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☑ Open Source BDM

○ Choose an Existing System

Connect to the USB-based Freescale Open Source BDM Cable.

< Back
Next >
Finish
Cancel

7. Click **Finish**.

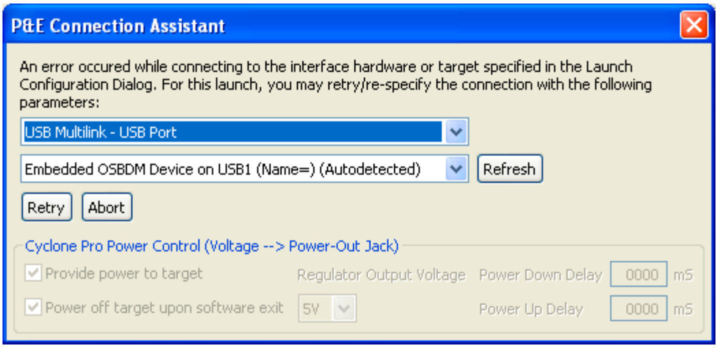
The wizard creates a project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select P&E Open Source BDM as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click Refresh. Refer to Figure 10.75.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 10.75 HCS08 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

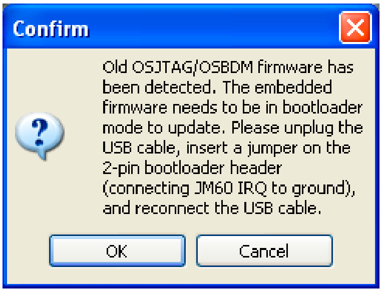
Embedded OSBDM Device on USB1 (Name=) (Autodetected)

Refresh
Retry
Abort
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage
5V
Power Down Delay
0000
mS
Power Up Delay
0000
mS

OSBDM Firmware Update

All Codewarrior IDE’s version 10.1 and higher have an automatic firmware update mechanism for built-in OSBDM hardware interfaces. Whenever an OSBDM-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSBDM firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

Figure 10.76 Old OSBDM Firmware Detected

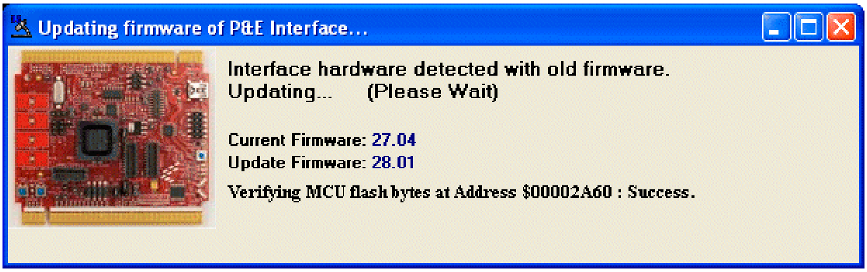


Confirm

Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSBDM device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSBDM-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSBDM device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSBDM device and click on the OK button. If done correctly, the automatic firmware update will occur.

Figure 10.77 OSBDM Firmware Updating



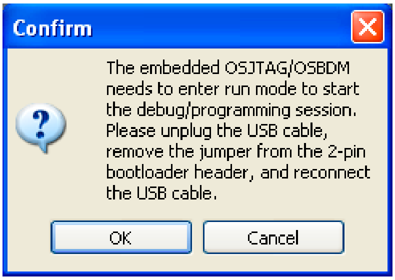
Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01

Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSBDM device must exit Bootloader mode and enter into Run mode.

Figure 10.78 Start OSBDM Run Mode



The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSBDM device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click on OK and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSBDM firmware. If for any reason you experience difficulty performing OSBDM firmware update, please visit www.pemicro.com/osbdm and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

11

Connections — RS08

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the RS08 full chip simulator or the target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the **Launch Configuration Settings** panel. **Launch Configuration Settings** panel can be accessed by clicking on the **Edit** button located within the **Main** tab of the **Debug Configurations** dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with simulation platforms and hardware devices that are part of the RS08 device family.

The topics in this chapter are:

- Changing Connection in IDE
- P&E Full Chip Simulation
- P&E RS08 Multilink/Cyclone PRO/OSBDM

Changing Connection in IDE

Full Chip Simulation (FCS) connection runs a complete simulation of all processor peripherals and I/O on your personal computer. Thus, when debugging an FCS project for a selected derivative it is not necessary to connect your PC with a Microcontrollers development or target board.

To select Full Chip Simulation as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **RS08** tree control and select the derivative or board you would like to use. For example, select `RS08 > RS08KA Family > MC9RS08KA1`.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Full Chip Simulation** checkbox.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project for the HCS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

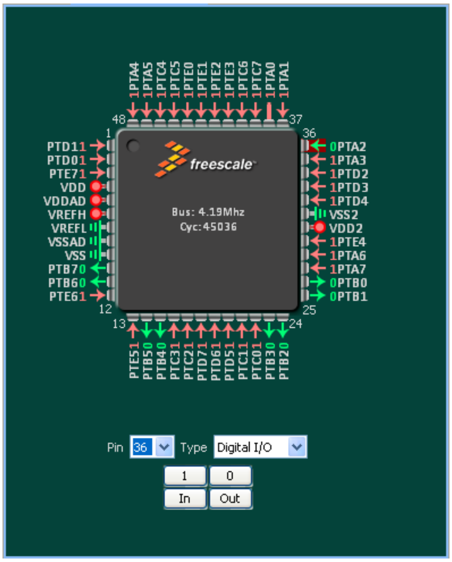
P&E Full Chip Simulation

This topic explains Chip View, which is a time-saving FCS feature, and also describes the settings of the connections that interface the CodeWarrior debugger with the RS08 simulator.

Chip View

Chip View is an innovative feature designed to simplify Full Chip Simulation (FCS) and In-Circuit Debugging (ICD) sessions. The **Chip View** provides instantaneous access to internal modules of the chip and lets you instantly change any of the features by clicking them. Each pin features the current pin direction, input/output value, and the name of the signal that reflects the current module that controls it. These data features are updated every 50ms throughout a running FCS or ICD session.

Figure 11.1 Chip View


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
freescale
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

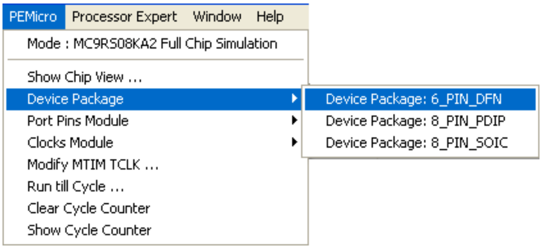
Bus: 4.19Mhz
Cyc:45036

Pin 36 Type Digital I/O

To open **Chip View**, perform these steps.

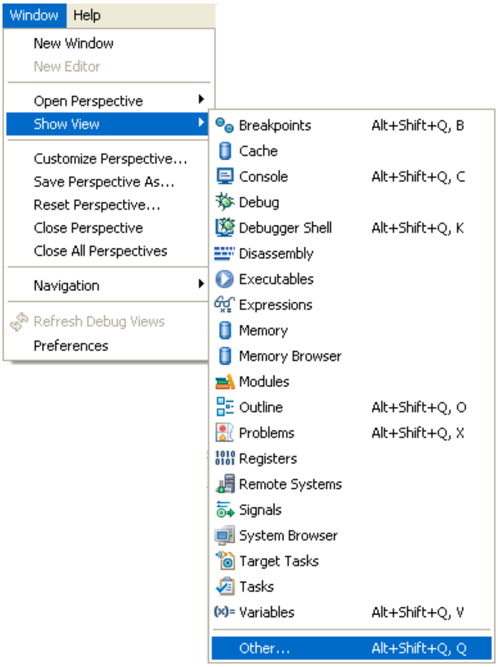
1. From the IDE menu bar, select **PEMicro > Device Package > Device Package:<** *Pin*>, where < *Pin*> is the pin package you would like to work with. (Refer Figure 11.2). The Device Package can be changed before or after the Chip View window is invoked within the CodeWarrior IDE.

Figure 11.2 Device Package Extended Menu


PEMicro
Processor Expert
Window
Help
Mode : MC9RS08KA2 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
Modify MTIM TCLK ...
Run till Cycle ...
Clear Cycle Counter
Show Cycle Counter
Device Package: 6_PIN_DFN
Device Package: 8_PIN_PDIP
Device Package: 8_PIN_SOIC

2. From the IDE menu bar, select **Window > Show View > Others** (Figure 11.3).

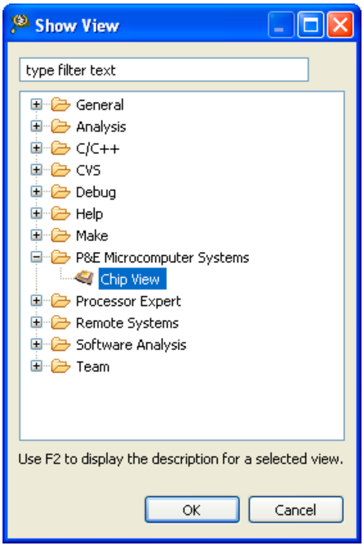
Figure 11.3 Show View Extended Menu


Window
Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Navigation
Refresh Debug Views
Preferences
Breakpoints Alt+Shift+Q, B
Cache
Console Alt+Shift+Q, C
Debug
Debugger Shell Alt+Shift+Q, K
Disassembly
Executables
Expressions
Memory
Memory Browser
Modules
Outline Alt+Shift+Q, O
Problems Alt+Shift+Q, X
Registers
Remote Systems
Signals
System Browser
Target Tasks
Tasks
Variables Alt+Shift+Q, V
Other... Alt+Shift+Q, Q

The **Show View** dialog box appears.

3. Expand **P&E Microcomputer Systems** and select **Chip View** (Figure 11.4).

Figure 11.4 Show View Menu


Show View
type filter text
General
Analysis
C/C++
CVS
Debug
Help
Make
P&E Microcomputer Systems
Chip View
Processor Expert
Remote Systems
Software Analysis
Team
OK
Cancel

Use F2 to display the description for a selected view.

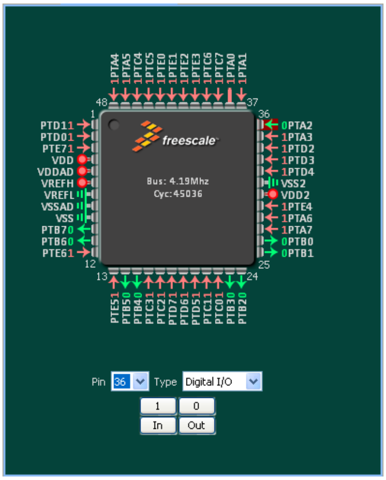
4. To change direction and values of the pin, double-click on the corresponding arrow or the number value. Details are listed below.

NOTE If you close the **Chip View** window during debug session, you will not be able to access to the Chip View.You must reopen the **Chip View** window and restart the current debugging session to open the **Chip View** window again. Closing **Chip View** should slightly improve performance during existing debug session.

Chip GUI - Ports Module Support

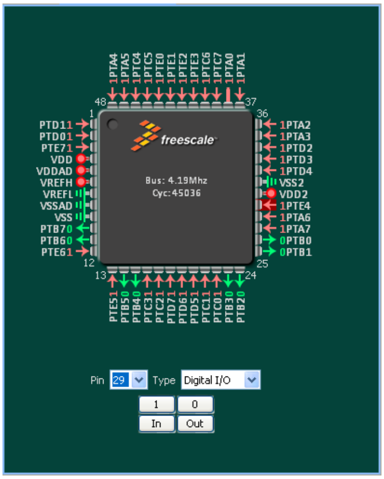
You have the option of changing the pin's direction and values by double-clicking on the corresponding arrow or number value. Figure 11.5 is an example of what the Chip View may look like before any changes are made. When the pin direction is input, the pin will display the current pin input value. When the pin direction is output, you have the option of double-clicking the number value to control the output value for the pin. Figure 11.6 is an example of the PTA2 pin value being changed from 0 to 1 by double-clicking on the number value.

Figure 11.5 Chip View Display Before Change


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
freescale
Bus: 4.19Mhz
Cyc:45036
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

Pin 36 Type Digital I/O

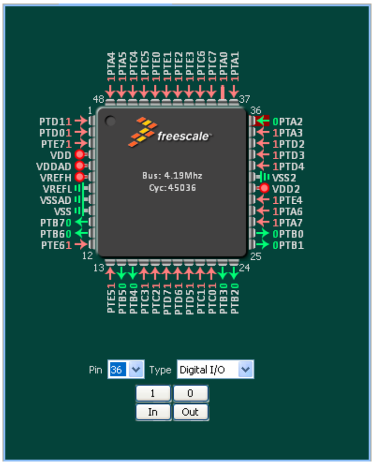
Figure 11.6 Chip View Display After Change


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
1PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
freescale
Bus: 4.19Mhz
Cyc:45036
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

Pin 29 Type Digital I/O

When you double-click a pin's value or direction, a pin configuration dialog box appears beneath the Chip View diagram (Figure 11.7). In the pin configuration options, you have the option of changing analog and digital I/O settings for a given pin. You can select a pin from the pin-number drop-down box, select between analog and digital signals, and switch pin directions. For the digital I/O signal, you can switch between high or low signals (Figure 11.7).

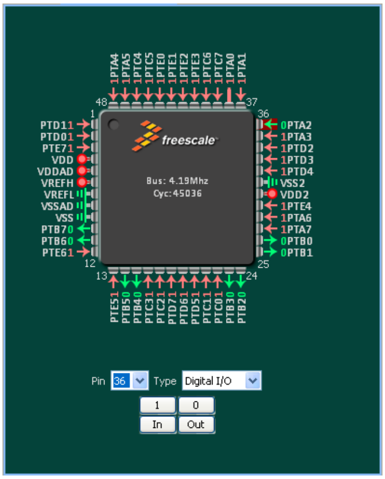
Figure 11.7 Chip View with Digital Pin Configuration Options


1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
freescale
Bus: 4.19Mhz
Cyc:45036
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

Pin 36 Type Digital I/O

For the Analog input signal, you can use the slider to change the analog signal value (Figure 11.8).

Figure 11.8 Chip View with Analog Pin Configuration Options

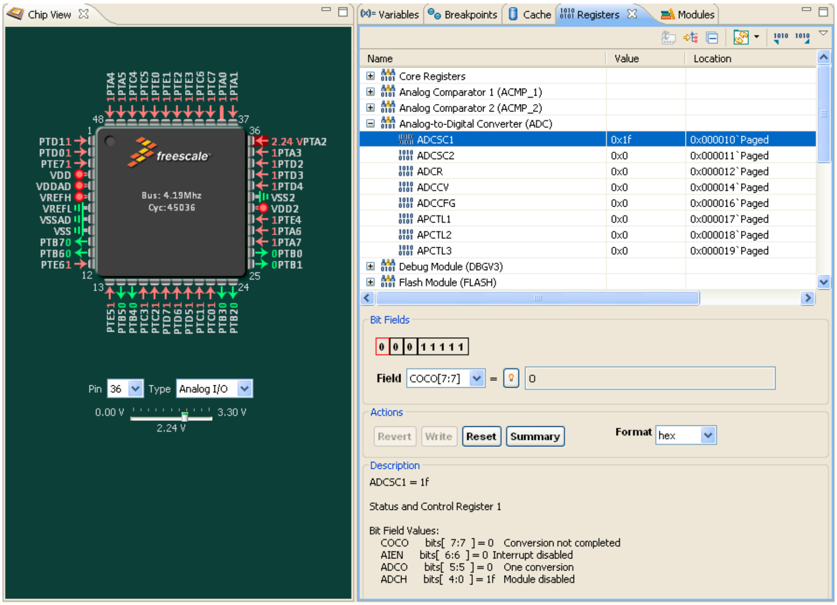

1PTA4
1PTA5
1PTC4
1PTC5
1PTE0
1PTE1
1PTE2
1PTE3
1PTC6
1PTC7
1PTA0
1PTA1
48
37
1
36
PTD11
PTD01
PTE71
VDD
VDDAD
VREFH
VREFL
VSSAD
VSS
PTB70
PTB60
PTE61
freescale
Bus: 4.19Mhz
Cyc:45036
0PTA2
1PTA3
1PTD2
1PTD3
1PTD4
VSS2
VDD2
1PTE4
1PTA6
1PTA7
0PTB0
0PTB1
12
25
13
24
PTE51
PTB50
PTB40
PTC31
PTC21
PTD71
PTD61
PTD51
PTC11
PTC01
PTB30
PTB20
1
0
In
Out

Pin 36 Type Digital I/O

Chip GUI - Analog to Digital Module Support

The Analog to Digital (ATD) Module has a higher priority than the General Pin I/O module. Therefore, if you have an ATD channel enabled and the ATD input buffer is empty, current input value on a pin will be converted and displayed in the ATD data conversion register (Figure 11.9).

Figure 11.9 Chip View with ATD


Chip View
PTA4 PTA5 PTC4 PTC5 PTE0 PTE1 PTE2 PTE3 PTC6 PTC7 PTA0 PTA1
48 37 1 36
PTD1 PTD0 PTE7 VDD VDDAD VREFH VREFL VSSAD VSS PTB7 PTB6 PTE6
2.24 VPTA2 PTA3 PTD2 PTD3 PTD4 VSS2 VDD2 PTE4 PTA6 PTA7 PTB0 PTB1
12 25 13 24
PTE5 PTB5 PTB4 PTC3 PTC2 PTD7 PTD6 PTD5 PTC1 PTC0 PTB3 PTB2
freescale
Bus: 4.19Mhz
Cyc:45036
Pin 36
Type Analog I/O
0.00 V
3.30 V
2.24 V
Variables
Breakpoints
Cache
Registers
Modules
Bit Fields
0 0 0 1 1 1 1 1
Actions
Revert
Write
Reset
Summary
Description

Name	Value	Location
Core Registers		
Analog Comparator 1 (ACMP_1)		
Analog Comparator 2 (ACMP_2)		
Analog-to-Digital Converter (ADC)		
ADCSC1	0x1f	0x000010`Paged
ADCSC2	0x0	0x000011`Paged
ADCR	0x0	0x000012`Paged
ADCCV	0x0	0x000014`Paged
ADCCFG	0x0	0x000016`Paged
APCTL1	0x0	0x000017`Paged
APCTL2	0x0	0x000018`Paged
APCTL3	0x0	0x000019`Paged
Debug Module (DBGV3)		
Flash Module (FLASH)		

Field COCO[7:7] = 0

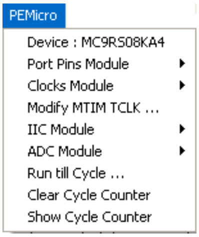
Format hex

```
ADCSC1 = 1f

Status and Control Register 1

Bit Field Values:
   COCO    bits[  7:7  ] = 0   Conversion not completed
   AIEN    bits[  6:6  ] = 0  Interrupt disabled
   ADCO    bits[  5:5  ] = 0   One conversion
   ADCH    bits[  4:0  ] = 1f   Module disabled
```

Module Options

The PEMicro menu (Figure 11.10) includes the Full Chip Simulation options for the modules that have specialty commands associated with them for a chosen device.

Figure 11.10 PEMicro Menu


PEMicro
Device : MC9RS08KA4
Port Pins Module
Clocks Module
Modify MTIM TCLK ...
IIC Module
ADC Module
Run till Cycle ...
Clear Cycle Counter
Show Cycle Counter

The options available are:

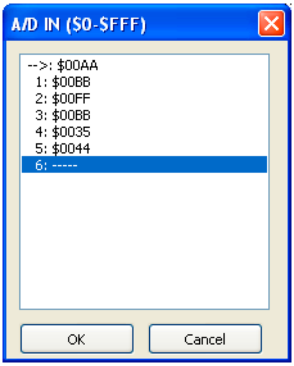
- ADC Module
- Internal Clock Source Module

- Inter-Integrated Circuit Module Option
- Keyboard Interrupt Module
- Liquid Crystal Display Driver Module Option
- Modulo Timer Interrupt Module
- Input/Output (I/O) Ports Module
- Serial Communications Interface Module
- Serial Peripheral Interface Module
- Timer Interface Module

ADC Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Analog to Digital Conversion (ADC) module including data input on all ADC channels, flag polling, interrupt operation, and the bus and CGMXCLK reference clock sources. FCS mode uses the buffered input structure to simulate the ADC inputs. You can queue up to 256 data values. To queue the ADC Input Data, use the ADDI command in the command prompt. If the data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is provided, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used as input to the ADC. The conversion takes place after a proper value is written to the ADC Status and Control register. Once the conversion occurs, the arrow moves to the next value in the ADC Buffer.

Figure 11.11 ADC IN Buffer Display

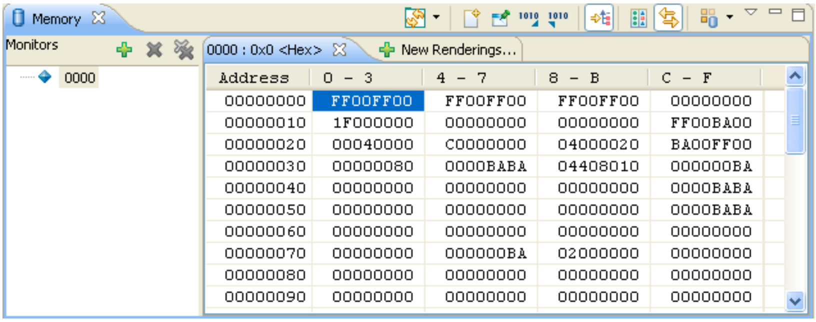


```
-->: $00AA
  1: $00BB
  2: $00FF
  3: $00BB
  4: $0035
  5: $0044
  6: -----
```

The ADCLR command can be used at any point to flush the input buffer for the ADC simulation.

After the conversion is complete, the first queued value is passed from the data buffer into the ADC data register. It can be observed in the Memory window by displaying the memory location corresponding to the ADC data register.

Figure 11.12 Memory Component Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

When the conversion is complete, FCS sets the appropriate flag. If interrupts are enabled, the Program Counter changes flow to the interrupt routine (as defined in the vector space of the MCU).

NOTE For more information on ADC configuration, refer to the Freescale user manual for your microprocessor.

ADC Module Commands

The following commands are available for the RS08/HCS08 ADC Module.

ADDI Command

The ADDI command lets you input the data into the ADC converter. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used by the ADC. The maximum number of input values is 256 bytes.

Syntax

```
>gdi ADDI [<n>]
```

Where:

`<n>` The value to be entered into the next location in the input buffer.

Example

```
>gdi ADDI $55
```

Set the next input value to the ADDI to $55

```
>gdi ADDI
```

Pull up the data window with all the input values.

ADCLR Command

Use the ADCLR command to flush the input buffer for ADC simulation. This resets the input data buffer and clears out all values. Notice that if the ADC is currently using a value, this command does not prevent the ADC from using it. Refer to ADDI command for information on how to access the input buffer of the ADC interface.

Syntax

```
>gdi ADCLR
```

Example

```
>gdi ADCLR
```

Clear the input buffer for ADC simulation.

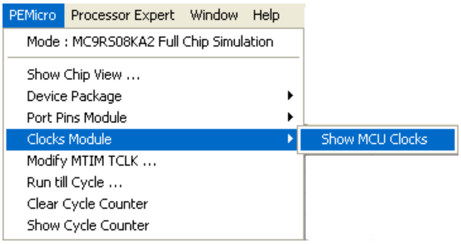
Internal Clock Source Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Internal Clock Source (ICS) Module, including:

- Phase Locked Loop (PLL) generation
- Automatic lock detection
- Interrupt
- Acquisition
- Tracking
- Flag polling

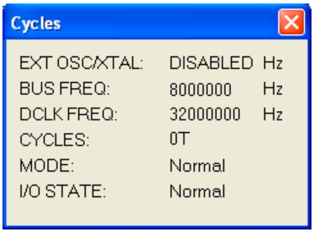
FCS mode uses a simulated External Oscillator Frequency change command (XTAL) lets you input the desired XTAL value. To check the current value of the External Oscillator, Bus Frequency and ICSCLK Frequency, open the RS08FCS menu and select Clocks Module > Show MCU Clocks.

Figure 11.13 Clocks Module Extended Menu


PEMicro
Processor Expert
Window
Help
Mode : MC9RS08KA2 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
Modify MTIM TCLK ...
Run till Cycle ...
Clear Cycle Counter
Show Cycle Counter
Show MCU Clocks

Once you select the MCU Clocks menu, the Cycles window displays all of the aforementioned Clock Frequencies, or you can select the Show Cycle Counter option within the FCS menu to get the same window.

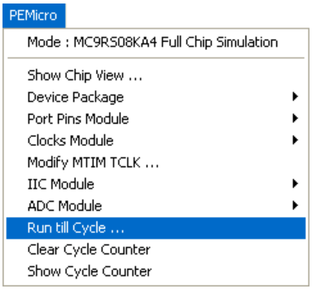
Figure 11.14 Frequency Display



EXT OSC/XTAL: DISABLED Hz
BUS FREQ: 8000000 Hz
DCLK FREQ: 32000000 Hz
CYCLES: 0T
MODE: Normal
I/O STATE: Normal

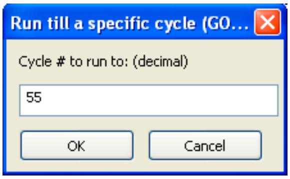
Within the FCS menu, you can select the Run till Cycle option, which lets you begin code execution and stop execution when the specified cycle count is reached. Note that the parameter given is not the number of cycles that executed, but rather the total cycle-count of the simulator (displayed in the Register Window).

Figure 11.15 Run till Cycle command


PEMicro
Mode : MC9RS08KA4 Full Chip Simulation
Show Chip View ...
Device Package
Port Pins Module
Clocks Module
Modify MTIM TCLK ...
IIC Module
ADC Module
Run till Cycle ...
Clear Cycle Counter
Show Cycle Counter

This command is extremely useful for verifying specific timings of a given event, running until a given event is complete, or just before it completes to enable stepping through the event or any application where cycle-timed execution is desired.

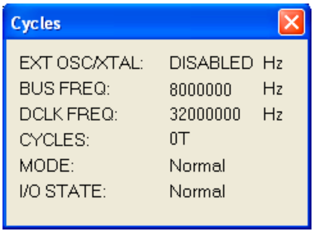
Figure 11.16 Run till Cycle Dialog Box



Cycle # to run to: (decimal)
55

You can also select the Clear Cycle Counter option within the FCS menu, which clears the cycle counter. If you select the Show Cycle Counter option within the FCS menu, you can check to make sure that the cycle counter is zero.

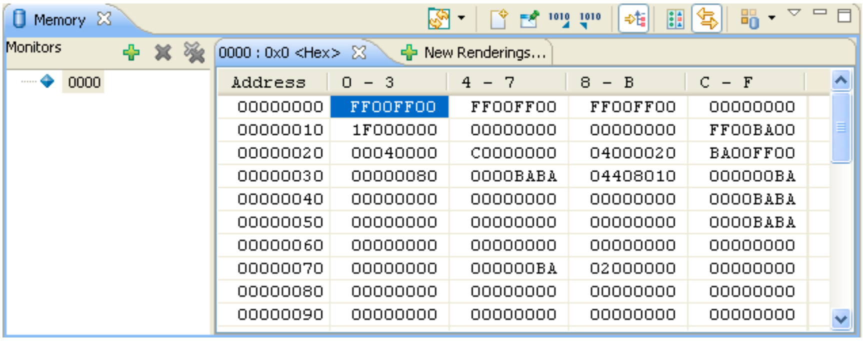
Figure 11.17 Cycle Counter Dialog Box with Cleared Counter



EXT OSC/XTAL:	DISABLED	Hz
BUS FREQ:	8000000	Hz
DCLK FREQ:	32000000	Hz
CYCLES:	0T	
MODE:	Normal	
I/O STATE:	Normal	

Once the ICG is properly configured, you can monitor the status of the PLL by polling the corresponding flag. If PLL interrupt is enabled, FCS jumps to an appropriate subroutine, as long as the interrupt vector is properly defined. To observe the flag going up as a result of the corresponding CPU event, situate your Memory window on the memory location of the ICG Status and Control register.

Figure 11.18 Memory Window



```
Address    0 - 3     4 - 7     8 - B     C - F
00000000   FF00FF00  FF00FF00  FF00FF00  00000000
00000010   1F000000  00000000  00000000  FF00BA00
00000020   00040000  C0000000  04000020  BA00FF00
00000030   00000080  0000BABA  04408010  000000BA
00000040   00000000  00000000  00000000  0000BABA
00000050   00000000  00000000  00000000  0000BABA
00000060   00000000  00000000  00000000  00000000
00000070   00000000  000000BA  02000000  00000000
00000080   00000000  00000000  00000000  00000000
00000090   00000000  00000000  00000000  00000000
```

For more information on how to properly configure Clock Generation, refer to the Freescale reference manual for your microprocessor.

Internal Clock Source Commands

The following commands are available for the RS08 Internal Clock Source Module.

XTAL Command

Use the XTAL command to change the value of the simulated external oscillator. This in turn affects the input to the PLL/DCO, and therefore the bus frequency. The P&E simulator is a cycle-based simulator, so changing the XTAL value does not affect the speed of simulation. It does, however, affect the ratio in which peripherals receive cycles. Certain peripherals that run directly from the XTAL will run at different speeds than those that run from the bus clock.

Syntax

```
>gdi XTAL <n>
```

Where:

• `<n>`, by default, is a hexadecimal number, representing the simulated frequency of an external oscillator. Adding the suffix `'t'` to the `'n'` parameter forces the input value to be interpreted as base 10.

Example

```
>gdi XTAL
```

Brings up an input window. The default base for this input value is 10. However, this value can be forced to a hexadecimal format through use of the suffix `'h'`.

Inter-Integrated Circuit Module Option

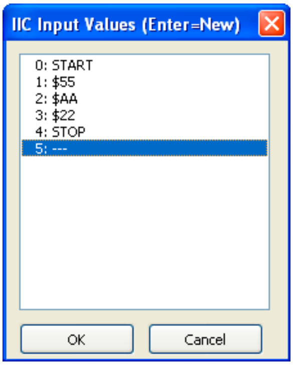
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Inter-Integrated Circuit (IIC) module including:

- Flag polling
- Interrupt enabled mode
- Transmission and reception of external data
- Master and slave modes of operation
- START and STOP signal generation detection
- Acknowledge bit generation detection

FCS mode uses the buffered input/output structure to simulate IIC inputs. You can queue up to 256 data bytes into the input buffer. The output buffer of the USB module can also hold 256 output bytes. To queue the IIC Input Packets, use the IICDI <...> command in the command prompt. For a more detailed description of the command, refer to the IIC Commands section. If the IIC packet parameters are properly defined, the packet is placed into the next slot in the input buffer. Otherwise, if no parameters are provided, an IIC Input Buffer window is displayed.

You can enter different IIC packet parameters while the window is open, including START, STOP, ACK, NACK and data bytes. An arrow points to the next byte to be used as input to the IIC. The data from the IIC input buffer is written to the IIC module registers once the IIC module is turned on and properly configured for receiving data from an external IIC device. Once simulation of the data transmission is over, the arrow moves to the next value in the IIC Input Buffer.

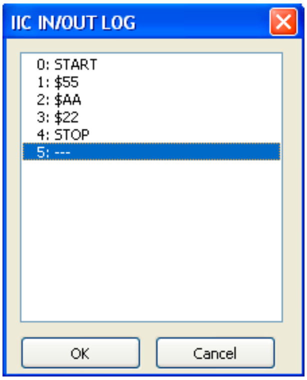
Figure 11.19 IIC Input Buffer Display


IIC Input Values (Enter=New)

0: START
1: $55
2: $AA
3: $22
4: STOP
5: ---

The IIC data input/output log buffer simulation lets you gain access to the past 256 IIC data bytes that have been shifted in and out of the module. To bring up the IIC IN/OUT LOG buffer dialog box, use the IICDO command.

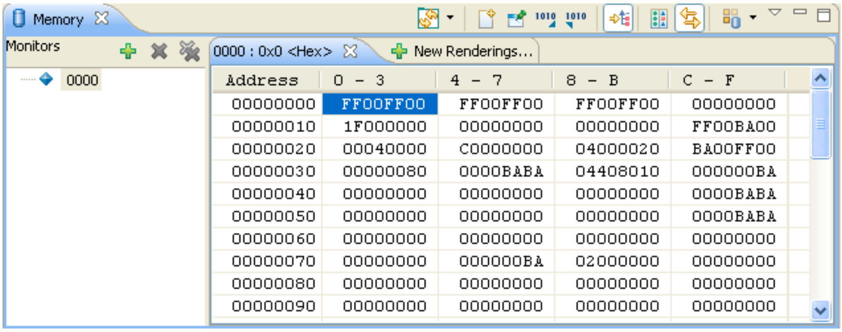
Figure 11.20 IIC IN/OUT LOG Buffer Display


IIC IN/OUT LOG

0: START
1: $55
2: $AA
3: $22
4: STOP
5: ---

The IICCLR command may be used at any point to flush the input as well as input/output log IIC buffers. After the IIC simulated input is received, the first queued-in data byte is passed from the data buffer into the corresponding IIC module registers. It can be observed in the Memory window by displaying the appropriate register location there.

Figure 11.21 Memory Component Window



```
Address    0 - 3     4 - 7     8 - B     C - F
00000000   FF00FF00  FF00FF00  FF00FF00  00000000
00000010   1F000000  00000000  00000000  FF00BA00
00000020   00040000  C0000000  04000020  BA00FF00
00000030   00000080  0000BABA  04408010  000000BA
00000040   00000000  00000000  00000000  0000BABA
00000050   00000000  00000000  00000000  0000BABA
00000060   00000000  00000000  00000000  00000000
00000070   00000000  000000BA  02000000  00000000
00000080   00000000  00000000  00000000  00000000
00000090   00000000  00000000  00000000  00000000
```

You can also observe different IIC flags in the Memory window. If you run the module in Flag Polling mode, poll the flag corresponding to the expected IIC event. If the IIC interrupts are enabled, FCS jumps to an appropriate subroutine as long as the IIC interrupt vectors are properly defined.

NOTE For more information on how to configure IIC module for desired operation, refer to the Freescale user manual for your microprocessor.

Inter-Integrated Circuit Module Commands

The following commands are available for the RS08 Inter-Integrated Circuit (IIC) module. Command function is identical even though the module names differ.

IICDI Command

The IICDI command lets you input data into a buffer of data to shift into the IIC module when it receives data from an external device. If a data parameter is given, the value is placed into the next slot in the input buffer. Otherwise, if no parameter is given, a window is displayed with the input buffer values. Input values can be entered while the window is open. The maximum number of input values is 256. This command is useful for either inputting response data from a slave target or for inputting data packets from an external master. Note that when the microprocessor attempts to read an acknowledge from an external device, and the next value in the buffer is neither ACK nor NACK, the microprocessor automatically receives an ACK signal (i.e. assumes ACK unless NACK is specified).

Syntax

```
>gdi IICDI [<n>][START][STOP][ACK][NACK]
```

Where:

- `<n>` indicates the value to be entered into the next location in the input buffer
- `START` indicates the incoming START signal
- `STOP` indicates the incoming STOP signal
- `ACK` corresponds to ACK signal
- `NACK` corresponds to NACK signal

NOTE For a detailed description of the IIC protocol and a proper way to configure the IIC module, refer to the Freescale user manual for your microprocessor.

Example

```
>gdi IICDI
```

Pulls up the data window with all the input values

```
>gdi IICDI 22 33
```

This is an example of data being returned from a slave device. Once the MCU transmits a start signal and the target address, it receives an ACK from the slave device. An ACK is implied unless a NACK is specified via the IICDI command. The next two data bytes read are 22 and 23. If the microprocessor attempts to read another byte, it gets an $FF value followed by a NACK signal (NACK because nothing remains in the input buffer). The receiving device then generates a STOP signal. A more exact input from a device designed to return two bytes is:

```
>gdi IICDI ACK 22 ACK 23 NACK
```

IIC in master mode transmits to a slave:

- If the slave device acknowledges all output bytes of the transmitting device, there is no need to specify an input packet. If the master device is going to transmit an address and two bytes, the following packet is equivalent to no packet:

```
>gdi IICDI ACK ACK ACK
```

- If, however, the slave receiver is designed to generate a NACK signal after the second received data byte, the proper response packet is:

```
>gdi IICDI ACK ACK NACK
```

- The address result being the first ACK, the first data result being the second ACK, and the second data byte being the NACK.

IIC in MASTER mode is not acknowledged by any Slave:

```
>gdi IICDI NACK
```

- If the NACK signal is entered before the master device transmits a START signal, then the master device gets a NACK when it tries to read an acknowledge after the address is output. The master device then generates a STOP signal and releases the BUS.

IIC in SLAVE mode receives a Write from an external Master:

This example is for an external master that is writing to the microprocessor configured to simulate the slave mode operation. The packet contains both START and STOP signals which puts the simulated device into the slave mode.

```
>gdi IICDI START 55 AA 22 STOP
```

This input adds five values to the input queue, which is a packet from an external master, including the following procedure values:

- A start signal comes in
- The address $55 comes in, specifying a write (slave receive). The Address Register in the current simulated device has been previously set to $55
- The data byte $AA comes in
- The data byte $22 comes in
- A STOP signal comes in

IICDO Command

The IICDO command displays a window, which shows data shifted in as well as shifted out of the IIC peripheral. An arrow points to the last output value transmitted/received. The maximum number of output values that the buffer can hold is 256.

Syntax

```
>gdi IICDO
```

Example

```
>gdi IICDO
```

View data from the input/output log buffer for IIC simulation.

IICCLR Command

Use the IICCLR command to flush the input and output buffers for IIC simulation. This resets the buffers and clears all values. Notice that if the IIC is currently shifting a value, this command does not prevent the IIC from finishing the transfer.

Syntax

```
>gdi IICCLR
```

Example

```
>gdi IICCLR
```

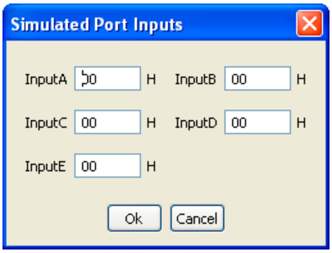
Clear input and output buffers for IIC simulation.

Keyboard Interrupt Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Keyboard Interrupt (KBI) module, including the edge-only, edge and level interrupt, and flag polling modes of operation. FCS mode uses simulated port inputs to trigger the KBI event from the proper I/O port pin.

To define an input state of the specific port, enter the INPUT<x> <n> command in the Command window. The <x> represents the corresponding I/O port, while <n> stands for the input value to write to this port. At the same time, you can use the INPUTS command to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Refer the documentation for Timer Module Commands for more information about the various forms of this command.

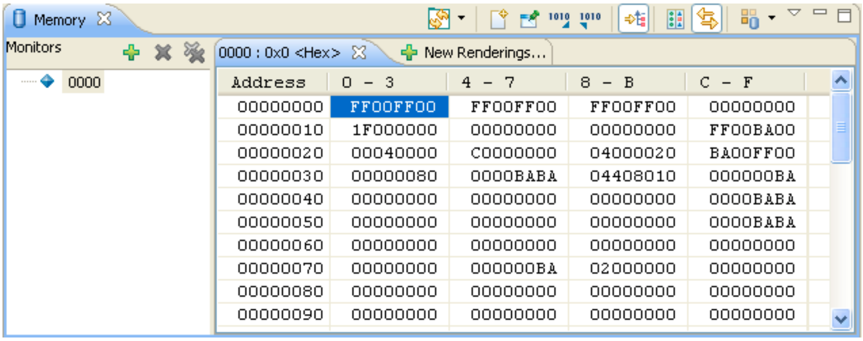
Figure 11.22 Simulated Port Inputs Dialog Box



InputA 00 H
InputB 00 H
InputC 00 H
InputD 00 H
InputE 00 H

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. To trigger the event, manipulate the inputs to the port in the appropriate manner, depending on whether the KBI is configured for edge-only or edge and level. Once the KBI event takes place, you can observe the KEYF Flag bit, which is a part of the Keyboard Status and Control register, in the Memory window.

Figure 11.23 Memory Component Window



```
Address    0 - 3      4 - 7      8 - B      C - F
00000000   FF00FF00   FF00FF00   FF00FF00   00000000
00000010   1F000000   00000000   00000000   FF00BA00
00000020   00040000   C0000000   04000020   BA00FF00
00000030   00000080   0000BABA   04408010   000000BA
00000040   00000000   00000000   00000000   0000BABA
00000050   00000000   00000000   00000000   0000BABA
00000060   00000000   00000000   00000000   00000000
00000070   00000000   000000BA   02000000   00000000
00000080   00000000   00000000   00000000   00000000
00000090   00000000   00000000   00000000   00000000
```

You can poll the KBI Interrupt Pending flag if the Polling Mode is simulated. In Interrupt Mode, the simulator branches to an appropriate interrupt subroutine as long as the KBI interrupt vector is properly configured.

NOTE For more information on KBI configuration, refer to the Freescale user manual for your microprocessor.

Keyboard Interrupt Commands

Use the following commands for Keyboard interrupt manipulation.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

`<x>` is the letter representing corresponding port

`<n>` is an eight-bit simulated value for port <x>

Example

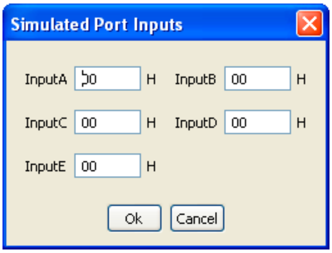
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation mode, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.24. You may then use this box to specify the input states of port pins and IRQ.

Figure 11.24 Simulated Port Inputs Dialog Box



InputA 00 H
InputB 00 H
InputC 00 H
InputD 00 H
InputE 00 H

When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

Liquid Crystal Display Driver Module Option

In Full Chip Simulation (FCS) Mode, this option lets you simulate all the functionality of the Liquid Crystal Display (LCD) module, including programmable LCD frame frequency, front plane pin configuration, back plane pin configuration, programmable blink frequency, and LCD interrupt flag generation. By default LCD front and back plane pins are mapped to match device use on the corresponding Freescale DEMO9RS08xx device board. These settings can be changed by you through modification of the LCDRS08V<x>_<DEVICE>.INI file, where <x> indicates the version number. This file is located in the "`<CW_Install>\prog\P&E`" folder.

Modulo Timer Interrupt Module

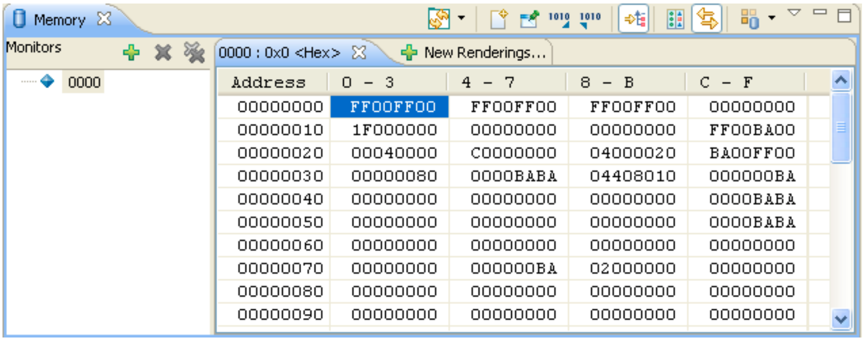
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Modulo Timer Interrupt (MTIM) Module, including:

- Programmable MTIM clock input
- Free running or modulo up count operation

- Flag polling
- Interrupt enabled mode of operation

Once the MTIM Status and Control register properly configures the operation of the module, the MTIM Counter starts incrementing. If modulo up count operation is enabled, you can observe the MTIM overflow flag in the MTIM Status and Control register in the Memory window.

Figure 11.25 Memory Component Window



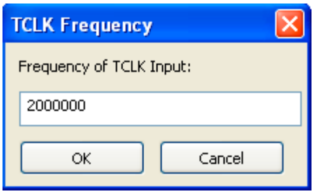
```
Address    0 - 3      4 - 7      8 - B      C - F
00000000   FF00FF00   FF00FF00   FF00FF00   00000000
00000010   1F000000   00000000   00000000   FF00BA00
00000020   00040000   C0000000   04000020   BA00FF00
00000030   00000080   0000BABA   04408010   000000BA
00000040   00000000   00000000   00000000   0000BABA
00000050   00000000   00000000   00000000   0000BABA
00000060   00000000   00000000   00000000   00000000
00000070   00000000   000000BA   02000000   00000000
00000080   00000000   00000000   00000000   00000000
00000090   00000000   00000000   00000000   00000000
```

If the MTIM interrupt is enabled, the FCS jumps to an appropriate subroutine as long as the MTIM interrupt vector is properly defined.

Modify MTIM TCLK

Figure 11.26 shows the TCLK frequency dialog box.

Figure 11.26 TCLK Frequency Dialog Box



Frequency of TCLK Input: 2000000

This dialog box lets you set the frequency of the TCLK signal for the MTIM peripheral. In order for this value to have any effect, the TCLK must be selected as the clock source for the MTIM.

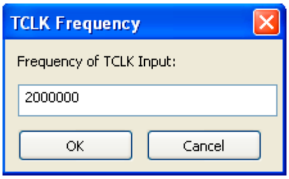
Modulo Timer Interrupt Module User Commands

The following commands are available for the MTIM.

TCLK Command

The TCLK opens the TCLK frequency dialog box shown in Figure 11.27. You may then use this box to specify the input frequency of the TCLK.

Figure 11.27 TCLK Frequency Dialog Box



Frequency of TCLK Input: 2000000

Syntax

```
>gdi TCLK
```

Example

```
>gdi TCLK
```

Show TCLK Frequency Dialog Box

TCLK <n> Command

The TCLK <n> command sets the TCLK input frequency to <n>.

Syntax

```
>gdi TCLK <n>
```

Where:

`<n>` is the input frequency of TCLK

Example

```
>gdi TCLK 200000t
```

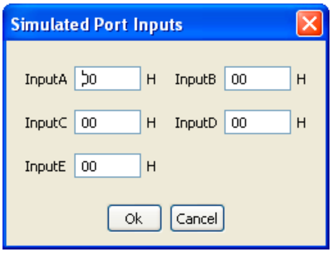
Simulate the TCLK input frequency of 200000 Hz.

Input/Output (I/O) Ports Module

In Full Chip Simulation (FCS) mode, this module simulates all input and output functionality of the Input/Output (I/O) Ports module. FCS mode uses a set of designated commands to simulate the input and output activity on corresponding I/O port pins. To define an input state of the specific port, write the INPUT <x> <n> command in the Command window. The <x> represents the corresponding I/O port, while the <n> stands for the input value to write to this port. At the same time, you can use the INPUTS command to bring up the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports.

NOTE Refer Input/Output Ports User Commands and IRQ Commands for more information about the various forms of this command.

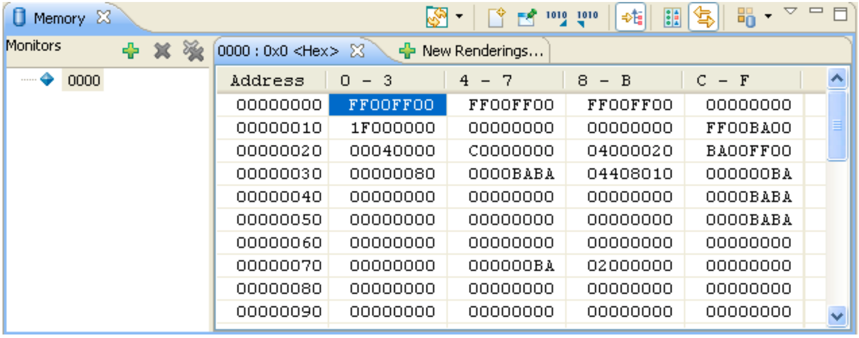
Figure 11.28 Simulated Port Inputs Dialog Box



InputA 00 H
InputB 00 H
InputC 00 H
InputD 00 H
InputE 00 H

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. Use the INPUTS command to reconfigure the output values on any relevant I/O port. You can observe the manipulation of I/O port pins in the Memory window.

Figure 11.29 Memory Component Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

Note that if the regular I/O pins are multiplexed to be used by a different MCU Module, they might not be available for general I/O functionality.

NOTE For more information on how to properly configure I/O pins, refer to the Freescale user manual for your microprocessor.

Input/Output Ports User Commands

Use the following commands for general I/O ports manipulation.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

<x> is the letter representing corresponding port

<n> Eight-bit simulated value for port <x>

Example

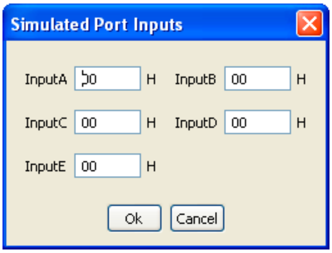
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation modes, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.30. You may then use this box to specify the input states of port pins and IRQ.

Figure 11.30 Simulated Port Inputs Dialog Box



InputA 00 H
InputB 00 H
InputC 00 H
InputD 00 H
InputE 00 H

When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

Serial Communications Interface Module

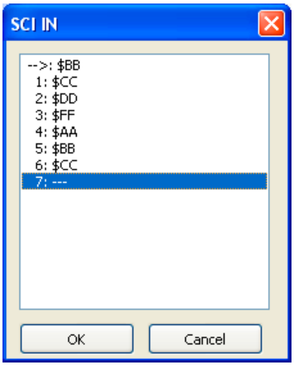
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- 8- or 9-bit length data codes
- Odd and even parity modes
- Transmission and reception of external data

FCS mode uses the buffered input/output structure to simulate SCI inputs. You can queue up to 256 data values into the input buffer. The output buffer of the SCI module can also hold 256 output values. To queue the SCI Input Data, use the SCDI <n> command in the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer.

Otherwise, if no parameter is provided, a window is displayed with the input buffer values. You can enter input values while the window is open. An arrow points to the next value to be used as input to the SCI. The data from the SCI input buffer is written to the SCI data register once the SCI module has been turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SCI IN Buffer.

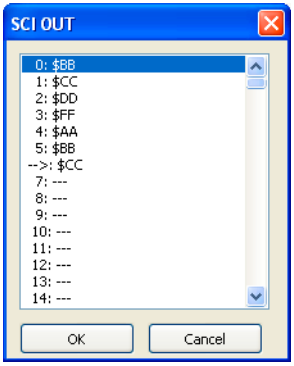
Figure 11.31 SCI IN Buffer Display



```
-->: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
  6: $CC
  7: ---
```

SCI Data Output Buffer simulation lets you gain access to the past 256 SCI data values transmitted out of the module. To bring up the SCI OUT buffer dialog box, use the SCDO command.

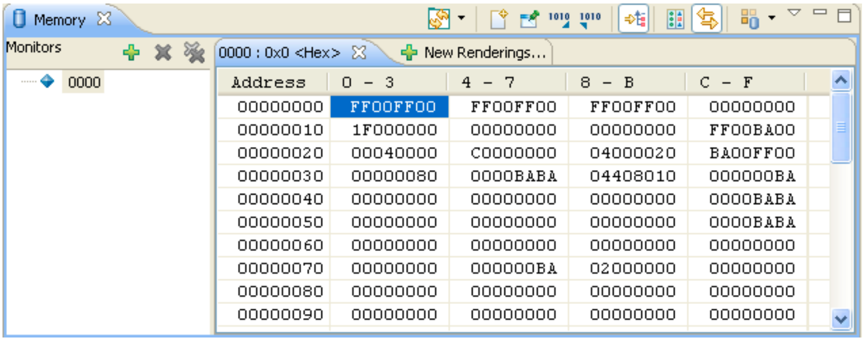
Figure 11.32 SCI OUT Buffer Display



```
0: $BB
1: $CC
2: $DD
3: $FF
4: $AA
5: $BB
-->: $CC
7: ---
8: ---
9: ---
10: ---
11: ---
12: ---
13: ---
14: ---
```

At any point, the SCCLR command may be used to flush the input and output SCI buffers. After the SCI simulated input is received, the first queued value is passed from the data buffer into the SCI data register. It can be observed in the memory window by displaying the memory location corresponding to the SCI data register.

Figure 11.33 Memory Component Window



```
Address   0 - 3     4 - 7     8 - B     C - F
00000000  FF00FF00  FF00FF00  FF00FF00  00000000
00000010  1F000000  00000000  00000000  FF00BA00
00000020  00040000  C0000000  04000020  BA00FF00
00000030  00000080  0000BABA  04408010  000000BA
00000040  00000000  00000000  00000000  0000BABA
00000050  00000000  00000000  00000000  0000BABA
00000060  00000000  00000000  00000000  00000000
00000070  00000000  000000BA  02000000  00000000
00000080  00000000  00000000  00000000  00000000
00000090  00000000  00000000  00000000  00000000
```

You can also observe different SCI flags in the Memory window. If the module is run in Flag Polling mode, poll the flag corresponding to the expected SCI event. If the SCI interrupts are enabled, the FCS jumps to an appropriate subroutine as long as the SCI interrupt vectors are properly defined.

NOTE For more information on how to configure the SCI module for desired operation, refer to the Freescale user manual for your microprocessor.

SCI Commands

Use the following commands for serial communication interface manipulation.

SCCLR Command

Use the SCCLR command to flush the input and output buffers for SCI simulation. This resets the buffers and clears out all values. Note that if the SCI is in the process of shifting a value, this command allows the SCI to finish the transfer. Refer the SCDI and SCDO commands for accessing the input and output buffers of the SCI interface.

Syntax

```
>gdi SCCLR
```

Example

```
>gdi SCCLR
```

Clear input and output buffer for SCI simulation

SCDI Command

The SCDI command lets you input data into the SCI. If a data parameter is given, the value is placed into the next slot in the SCI input buffer. If no parameter is given, a window displays the input buffer values. Input values can be entered while the window is open. An arrow points to the next value to be used as input to the SCI. The maximum number of input values is 256 bytes.

Syntax

```
>gdi SCDI [<n>]
```

Where:

`<n>` The value to be entered into the next location in the input buffer

Example

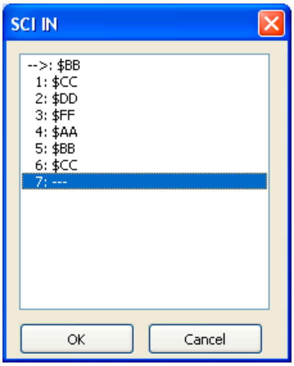
```
>gdi SCDI $55
```

Set the next input value to the SCI to $55

```
>gdi SCDI
```

Pull up the data window with all the input values.

Figure 11.34 SCI IN buffer display



```
-->: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
  6: $CC
  7: ---
```

SCDO Command

The SCDO command displays the output buffer from the SCI. A window is opened that shows all the data that the SCI has shifted out. An arrow points to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax

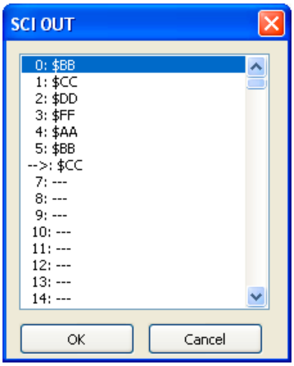
```
>gdi SCDO
```

Example

```
>gdi SCDO
```

View data from the output buffer for the SCI simulation.

Figure 11.35 SCI OUT Buffer Display



```
  0: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
-->: $CC
  7: ---
  8: ---
  9: ---
 10: ---
 11: ---
 12: ---
 13: ---
 14: ---
```

Serial Peripheral Interface Module

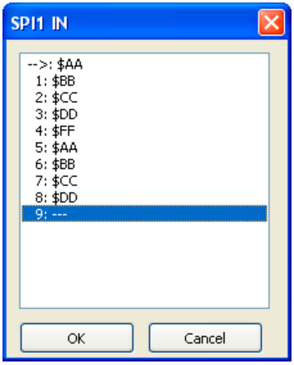
In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Serial Peripheral Interface (SPI) module including:

- Flag polling
- Interrupt enabled mode
- Master and slave modes
- Slave input clock
- Transmission and reception of external data

FCS mode uses the buffered input/output structure to simulate SPI inputs. You can queue up to 256 data values into the input buffer. The output buffer of the SPI module can also hold 256 output values. To queue the SPI Input Data, use the SPDI <n> command at the command prompt. If <n> (the data parameter) is given, the value is placed into the next slot in the input buffer.

Otherwise a window is displayed with the input buffer values. You can enter input values while the window is open. An arrow points to the next input value to the SPI. The data from the SPI input buffer is written to the SPI data register once the SPI module is turned on and is properly configured for receiving data from an external serial device. Once the simulation of the data transmission is over, the arrow moves to the next value in the SPI IN Buffer.

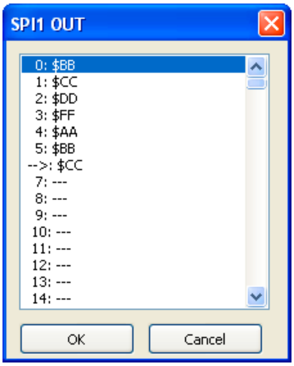
Figure 11.36 SPI IN Buffer Display



```
-->: $AA
  1: $BB
  2: $CC
  3: $DD
  4: $FF
  5: $AA
  6: $BB
  7: $CC
  8: $DD
  9: ---
```

SPI data output buffer simulation lets you gain access to the past 256 SPI data values transmitted out of the module. To bring up the SPI OUT buffer dialog box, use the SPDO command.

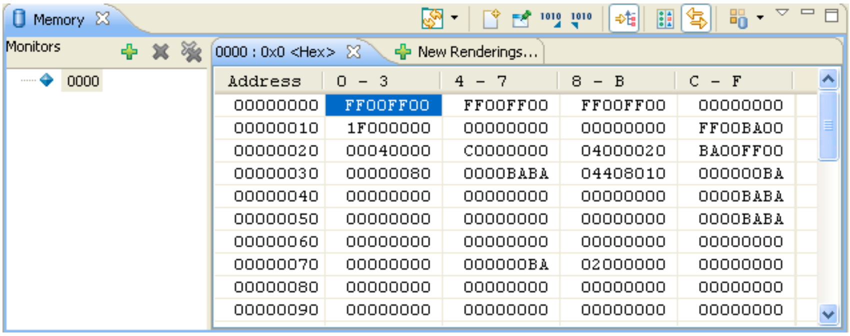
Figure 11.37 SPI OUT Buffer Display



```
0: $BB
1: $CC
2: $DD
3: $FF
4: $AA
5: $BB
-->: $CC
7: ---
8: ---
9: ---
10: ---
11: ---
12: ---
13: ---
14: ---
```

The SPCLR command may be used at any point to flush the input and output SPI buffers. After the SPI simulated input is received, the first queued value is passed from the data buffer into the SPI data register. It can be observed in the Memory window by displaying the memory location corresponding to the SPI data register.

Figure 11.38 Memory Component Window



```
Address    0 - 3     4 - 7     8 - B     C - F
00000000   FF00FF00  FF00FF00  FF00FF00  00000000
00000010   1F000000  00000000  00000000  FF00BA00
00000020   00040000  C0000000  04000020  BA00FF00
00000030   00000080  0000BABA  04408010  000000BA
00000040   00000000  00000000  00000000  0000BABA
00000050   00000000  00000000  00000000  0000BABA
00000060   00000000  00000000  00000000  00000000
00000070   00000000  000000BA  02000000  00000000
00000080   00000000  00000000  00000000  00000000
00000090   00000000  00000000  00000000  00000000
```

You can also observe different SPI flags in the Memory window. If the module is run in Flag Polling mode, poll the flag corresponding to the expected SPI event. If the SPI interrupts are enabled, the FCS jumps to an appropriate subroutine as long as the SPI channel interrupt vectors are properly defined.

To simulate the frequency of the SPI slave input clock, use the SPFREQ <n> command. If the SPI is configured for slave mode, this command lets you enter the number of cycles <n> in the period of the input clock. If the SPFREQ command is not used, then clocking is set by the SPI control register.

NOTE For more information on how to configure the SPI module for desired operation, refer to the Freescale user manual for your microprocessor.

SPI Commands

The following serial peripheral interface commands are available for the RS08.

SPCLR Command

Use the SPCLR command to flush the input and output buffers for SPI simulation. This resets the buffers and clears out all values. Notice that if the SPI is currently shifting a value, this command allows the SPI to finish the transfer. Refer the SPDI and SPDO commands for accessing the input and output buffers of the SPI interface.

Syntax

```
>gdi SPCLR
```

Example

```
>gdi SPCLR
```

Clear input and output buffer for SPI simulation

SPDI Command

The SPDI command lets you input data into the SPI. If a data parameter is given, the value is placed into the next slot in the SPI input buffer. If no parameter is given, a window displays the input buffer values. You can enter input values while the window is open. An arrow points to the next input value to the SPI. The maximum number of input values is 256 bytes.

Syntax

```
>gdi SPDI [<n>]
```

Where:

<n> The value to be entered into the next location in the input buffer

Example

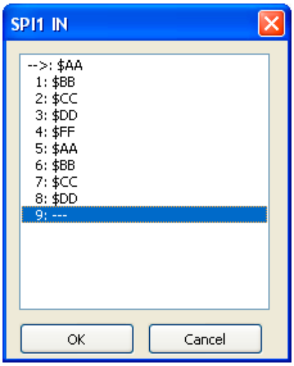
```
>gdi SPDI $55
```

Set the next input value to the SPI to $55

```
>gdi SPDI
```

Pull up the data window with all the input values.

Figure 11.39 SPI IN Buffer Display



```
-->: $AA
  1: $BB
  2: $CC
  3: $DD
  4: $FF
  5: $AA
  6: $BB
  7: $CC
  8: $DD
  9: ---
```

SPDO Command

The SPDO command displays the output buffer from the SPI. A window opens that shows all the data that the SPI has shifted out. An arrow points to the last output value transmitted. The maximum number of output values that the buffer holds is 256 bytes.

Syntax

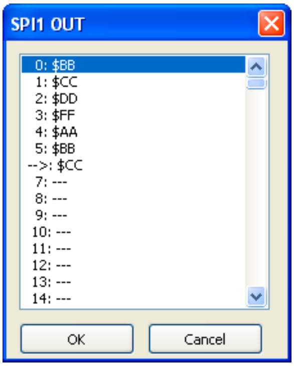
```
>gdi SPDO
```

Example

```
>gdi SPDO
```

View data from the output buffer for the SPI simulation.

Figure 11.40 SPI OUT Buffer Display



```
  0: $BB
  1: $CC
  2: $DD
  3: $FF
  4: $AA
  5: $BB
-->: $CC
  7: ---
  8: ---
  9: ---
 10: ---
 11: ---
 12: ---
 13: ---
 14: ---
```

SPFREQ Command

The SPFREQ command lets you set the frequency of the SPI slave input clock. If the SPI is configured for the slave mode, this command lets you enter the number of cycles <n> per one input clock period. If no value is given, a window appears and you are prompted for a value. If this command is not used, then the clocking is assumed to be set by the SPI control register.

Syntax

```
>gdi SPFREQ [<n>]
```

Where:

`<n>` The number of cycles for the period of the input clock.

Example

```
>gdi SPFREQ 8
```

Set the period of the input slave clock to 8 cycles (total shift = 8*8 cycles per bit = 64 cycles)

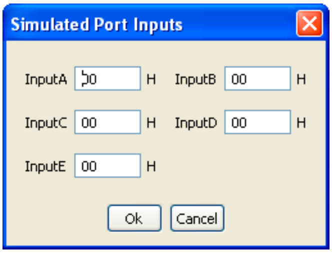
Timer Interface Module

In Full Chip Simulation (FCS) mode, this module simulates all functionality of the Timer Interface module, including:

- Input capture/output compare
- Pulse width modulation
- Internal or external clock input
- Free running or modulo up count operation
- Flag polling
- Interrupt enabled mode of operation

FCS mode uses the simulated port inputs to trigger the input capture on a given timer channel. To define an input state of the specific port, use the INPUT<x> <n> command in the Command window. The <x> represents the corresponding I/O port, while <n> stands for the input value to write to this port. At the same time, you can use the INPUTS command to display the Simulated Port Inputs for all general I/O ports. It displays the current simulated values to all applicable input ports. Refer the documentation for Timer Module Commands for more information about the various forms of this command.

Figure 11.41 Simulated Port Inputs Dialog Box



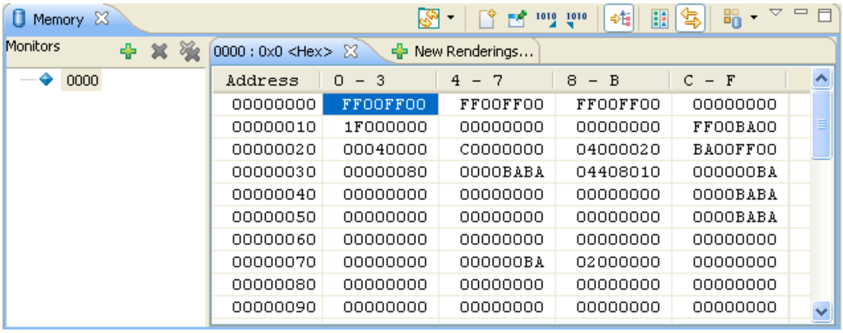
InputA 00 H InputB 00 H

InputC 00 H InputD 00 H

InputE 00 H

Use the Simulated Port Inputs dialog box to reconfigure the input value to any I/O port. To trigger the event, first set the port inputs high or low and then invert them to an opposite value, depending on whether the input capture is set for rising/falling edge. Once the Input Capture event takes place, you can observe the CHxF in the Channel Status and Control register in the Memory window.

Figure 11.42 Memory Component Window



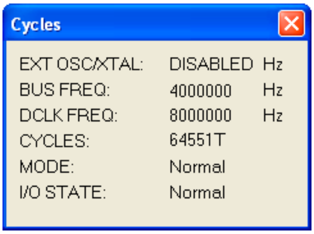
Address	0 - 3	4 - 7	8 - B	C - F
00000000	FF00FF00	FF00FF00	FF00FF00	00000000
00000010	1F000000	00000000	00000000	FF00BA00
00000020	00040000	C0000000	04000020	BA00FF00
00000030	00000080	0000BABA	04408010	000000BA
00000040	00000000	00000000	00000000	0000BABA
00000050	00000000	00000000	00000000	0000BABA
00000060	00000000	00000000	00000000	00000000
00000070	00000000	000000BA	02000000	00000000
00000080	00000000	00000000	00000000	00000000
00000090	00000000	00000000	00000000	00000000

If the Timer module is configured for an Output Compare event, then once the event takes place you can observe the same CHxF Flag via the Memory window. If the timer channel interrupt is enabled, the FCS jumps to an appropriate subroutine as long as the Timer channel interrupt vector is properly defined. To observe the Timer Overflow Flag (TOF) flag being set as a result of the corresponding CPU event, situate your Memory window on the memory location of the Timer Status and Control register.

To observe the Pulse Width Modulation (PWM) operation, properly configure the Timer to operate in the Modulo up count mode, then select the toggle-on-overflow or clear/set output on compare events to create the desired duty cycle wave. Once a PWM event takes place, you can observe pin toggle/clear/set behavior corresponding to the Timer configuration in the Memory window that is displaying the I/O port associated with a given timer channel.

To observe the accuracy of the Timer module operation, you can observe the number of CPU cycles that it takes for the event to occur. The cycle counter is only incremented as you step through the code. To determine the exact amount of cycles over which the event occurs, one can either observe the cycle display in the Register window or use the built in simulation commands. To display the current number of cycles in the Command window, use the CYCLES command. To change the number of cycles in the cycle counter, use CYCLES <n>, where <n> is the new cycle value. If the event has a pre-calculated number of cycles, use CYCLE 00 to reset the number of cycles and GOTOCYCLE <n> to run through the code until you reach the expected event.

Figure 11.43 Register Window With Cycles Display



EXT OSC/XTAL: DISABLED Hz
BUS FREQ: 4000000 Hz
DCLK FREQ: 8000000 Hz
CYCLES: 64551T
MODE: Normal
I/O STATE: Normal

Timer Module Commands

The following timer module commands are available for use with the HC08/HCS08 processors.

CYCLES Command

The CYCLES command changes the value of the cycles counter. The cycles counter counts the number of the processor cycles that have passed during execution. The Cycles Window shows the cycle counter. The cycle count can be useful for timing procedures.

Syntax

```
>gdi CYCLES <n>
```

Where:

`<n>` Integer value for the cycles counter

Examples

```
>gdi CYCLES 0
```

Reset cycles counter

```
>gdi CYCLES 1000
```

Set cycle counter to 1000.

GOTOCYCLE Command

The GOTOCYCLE command executes the program in the simulator beginning at the address in the program counter (PC). Execution continues until the cycle counter is equal to or greater than the specified value, until a key or the Stop button on the toolbar is pressed, until it reaches a break point, or until an error occurs.

Syntax

```
>gdi GOTOCYCLE <n>
```

Where:

`<n>` Cycle-counter value at which the execution stops

Example

```
>gdi GOTOCYCLE 100
```

Execute the program until the cycle counter equals 100.

INPUT<x> Command

The INPUT<x> command sets the simulated inputs to port <x>. The CPU reads this input value when port <x> is set as an input port.

Syntax

```
>gdi INPUT<x> <n>
```

Where:

`<x>` is the letter representing corresponding port

`<n>` Eight-bit simulated value for port <x>

Example

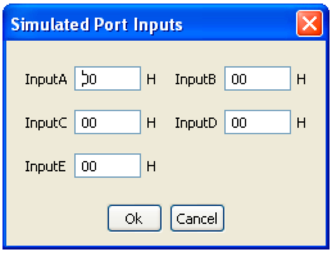
```
>gdi INPUTA AA
```

Simulate the input AA on port A.

INPUTS Command

In FCS and CPU-Only Simulation modes, the INPUTS command opens the Simulated Port Inputs dialog box shown in Figure 11.44. You may then use this box to specify the input states of port pins and IRQ.

Figure 11.44 Simulated Port Inputs Dialog Box



When using In-Circuit Simulation mode, the INPUTS command shows the simulated input values to any applicable port.

Syntax

```
>gdi INPUTS
```

Example

```
>gdi INPUTS
```

Show I/O port input values.

P&E RS08 Multilink/Cyclone PRO/OSBDM

This section describes the RS08 P&E Connection options. The RS08 Connection setting permits a connection to RS08 Freescale devices via P&E Multilink/Cyclone PRO/OSBDM hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

Connection Options

This topic describes all P&E RS08 Multilink/Cyclone Pro/OSBDM connection options, which are common to all P&E USB BDM Multilink or USB Multilink Universal, P&E Cyclone PRO Serial, P&E Cyclone PRO USB, P&E Cyclone PRO Ethernet, and P&E Open Source BDM connections.

The options include:

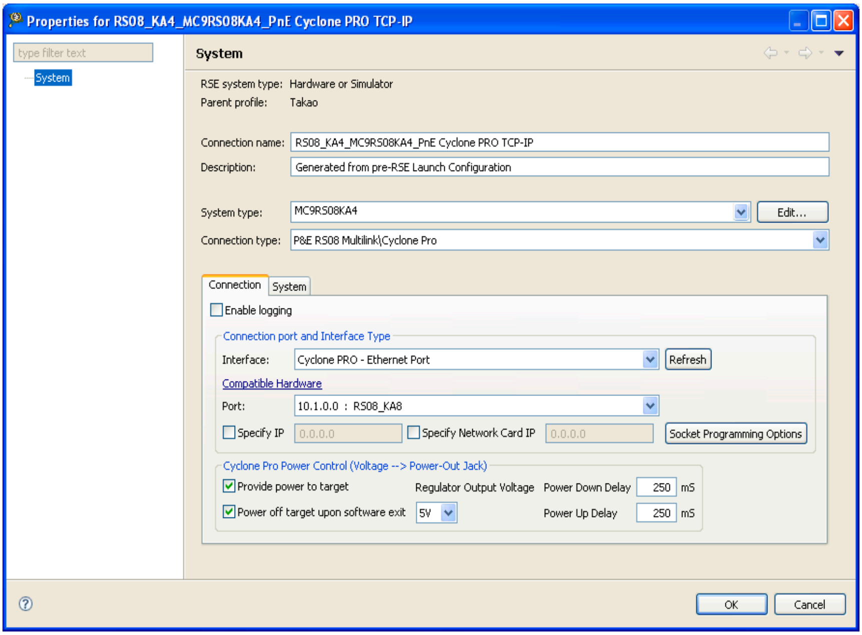
- Changing P&E Connection Settings
- Connection Assistant
- Launch Configuration Settings
- Active Mode Menu Options
- Advanced Programming/Debug Options

- View Register Files Option
- Socket Programming Options Button
- P&E RS08 Multilink/Cyclone PRO/OSBDM Connection-Specific Options

Changing P&E Connection Settings

All connection settings for P&E hardware interfaces are configured using the **Remote System** group in the **Main** tab of the **Debugger Configuration** dialog box.

Figure 11.45 P&E RS08 BDM Launch Configuration Dialog Box


Properties for RS08_KA4_MC9RS08KA4_PnE Cyclone PRO TCP-IP

RSE system type: Hardware or Simulator
Parent profile: Takao
Connection name: RS08_KA4_MC9RS08KA4_PnE Cyclone PRO TCP-IP
Description: Generated from pre-RSE Launch Configuration
System type: MC9RS08KA4
Connection type: P&E RS08 Multilink\Cyclone Pro
☐ Enable logging
Interface: Cyclone PRO - Ethernet Port
Port: 10.1.0.0 : RS08_KA8
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0
☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

type filter text
System
System
Edit...
Connection
System
Connection port and Interface Type
Refresh
Compatible Hardware
Socket Programming Options
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
OK
Cancel

Table 11.1 describes the options for this view.

WARNING! An improper voltage setting can damage the board.

Table 11.1 Connection Parameter Options for P&E RS08 Multilink/Cyclone PRO/OSBDM

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB HCS08\HCS12\CFV1 Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone PRO - Serial Port • Cyclone PRO - USB Port • Cyclone PRO - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project. • OSBDM
Refresh	Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE:** If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help.
Socket Programming Options	The Socket Programming Options button brings up a dialog that provides you with a graphical representation of the signals that must be connected from the BDM header to the pins of the microprocessor, in order to use Freescale socket adapters.

Table 11.1 Connection Parameter Options for P&E RS08 Multilink/Cyclone PRO/OSBDM

Option	Description
(Cyclone Ethernet only) Specify IP	Use this option to specify the IP address of a Cyclone outside of the local network. Click on the checkbox to enable the textbox. This will also disable the port dropdown box. Currently supports IPv4 only.
(Cyclone Ethernet only) Specify Network Card IP	Use this option to specify the local network card IP address if there are multiple cards on your computer. Click on the checkbox to enable the textbox. Currently supports IPv4 only.
(Cyclone PRO only) Provide power to target	This option determines whether the Cyclone PRO (circuitry) provides power to the target hardware via the probe. Check this option to have the Cyclone PRO (circuitry) supply power to the hardware target Uncheck this option to not provide power.
(Cyclone PRO only) Power off target upon software exit	This option determines whether Cyclone PRO hardware interface provides power to the target hardware via VDD of the BDM cable. Check this option to turn off the power when the program terminates. Uncheck this option to leave the hardware target powered continuously.
(Cyclone PRO only) Regulator Output Voltage	This option adjusts the output voltage that powers the hardware target. Select a voltage value from this option’s list box.

Table 11.1 Connection Parameter Options for P&E RS08 Multilink/Cyclone PRO/OSBDM

Option	Description
(Cyclone PRO only) Power down delay	This option specifies amount of time for which the target will be turned off during a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option's text box.
(Cyclone PRO only) Power up delay	This option specifies amount of time for which the target will remain powered prior to a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option's text box.

WARNING! An improper voltage setting can damage the board.

To change P&E Connections settings, perform these steps.

1. In the **CodeWarrior Projects** view, select the project for which you want to change the P&E Connections settings.

NOTE It is assumed that you have created a project and built it.

2. Select **Run > Debug Configurations** from the main menu bar of the IDE.

   The **Debug Configurations** dialog box appears.

3. Expand the **CodeWarrior Download** tree control in the left pane and select the launch configuration you want to debug.
4. Click the **Main** tab.

   The **Main** page appears in the area beneath the tabs.

5. Select a system within **Remote System** which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the Freescale Eclipse Extensions Guide. Once a remote system is selected, click the **Edit** button. The **Launch Configuration Settings Panel** will appear.
6. Ensure that the **System Type** is the correct MCU you want to debug. Use the drop-down box or the **Edit** button to change this option.
7. In the **Connection Type** drop-down box, select P&E RS08 Multilink\Cyclone Pro. The P&E connections settings will appear below.

8. Click **Refresh** to scan valid interface and port.

   Valid interfaces and ports appear in the **Interface** and **Port** drop-down lists in the **Connection** port and **Interface Type** group.

9. Select a supported interface from the **Interface** drop-down list.
10. Select a supported port from the **Port** drop-down list.

NOTE The port displayed may vary depending on the interface. For example, if you select interface as Cyclone PRO - Serial Port, the available port option is COM1 : Serial Port 1.

11. Specify settings in the **Cyclone Pro Power Control** (**Voltage --> Power -Out Jack**) group.

NOTE This group will be enabled for Cyclone PRO interface only.

- Check the **Provide power to target** checkbox to have the Cyclone PRO (circuitry) provide power to the target else clear the checkbox if you do not want to provide power to the target.
- Check the **Power off target upon software exit** checkbox to turn off the power when the program terminate else clear the checkbox to leave the hardware target powered continuously.
- Select a voltage value from the **Regulator Output Voltage** drop-down list. This adjusts the output voltage that powers the hardware target.

WARNING! An improper voltage setting can damage the board.

- Enter the delay interval (in milliseconds) in the **Power Down Delay** text box. This option specifies the time interval to wait before shutting off the power to the hardware target. Cyclone Pro powers down the device once the debug session is over, or while executing a power cycling sequence after beginning a new debug session.
- Enter the delay interval (in milliseconds) in the **Power Up Delay** text box. This option specifies the time interval to wait before turning on the power to the hardware target. If the Cyclone Pro power to target feature is enabled, Cyclone Pro will power up the device while executing a power cycling sequence at the beginning of every debug session.

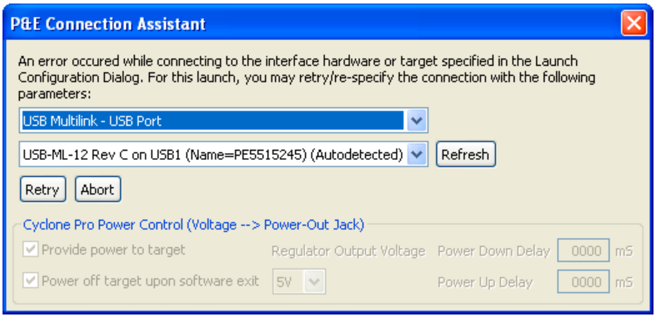
12. Click **OK** to save changes to the P&E Connections settings. The Launch Configuration Settings Panel dialog box will close.
13. Click **Close** button to close the Debug Configuration dialog box.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Multilink/Cyclone PRO/OSBDM** as your debugger connection:

1. Select the **P&E** device that you are using from the first drop-down menu and click **Refresh**. Refer Figure 11.46.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings (Cyclone Pro only). Refer to Table 11.1 for more details of each setting.
4. Click the Retry button

Figure 11.46 RS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

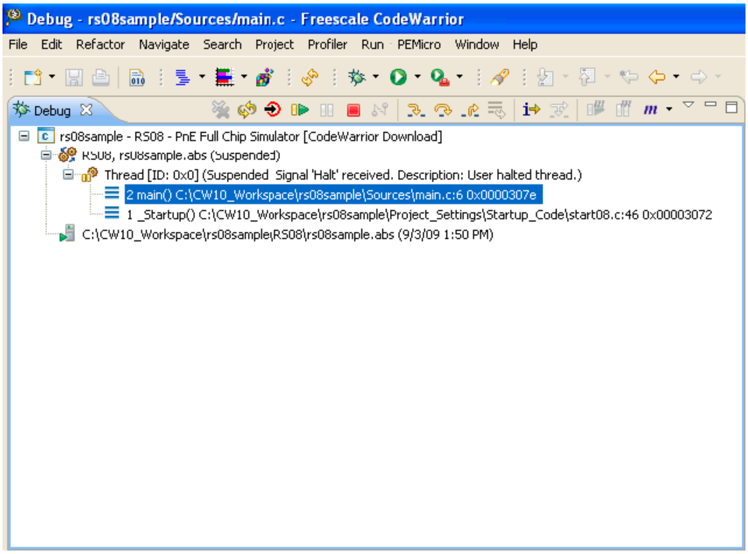
Power Up Delay: 0000 mS

Launch Configuration Settings

To set the launch configurations for the debugger:

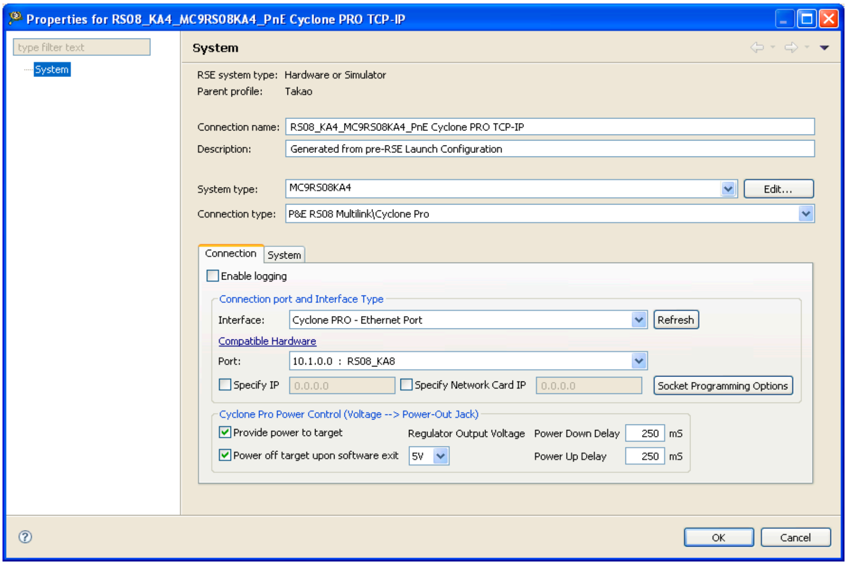
1. Find the debugger icon and click on the drop-down arrow to bring up the debugger menu. Refer Figure 11.47.
2. Select **Debug Configurations**.
3. On the left column, select the project download type you would like to set the launch configurations. Refer Figure 11.48
4. On the right column, click on the **Debugger** tab.
5. Set your configurations and click **Debug** to start the debugger.

Figure 11.47 Debugger Drop-down Menu


Debug - rs08sample/Sources/main.c - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run PEMicro Window Help
Debug

rs08sample - RS08 - PnE Full Chip Simulator [CodeWarrior Download]
RS08, rs08sample.abs (Suspended)
Thread [ID: 0x0] (Suspended Signal 'Halt' received. Description: User halted thread.)
2 main() C:\CW10_Workspace\rs08sample\Sources\main.c:6 0x0000307e
1 _Startup() C:\CW10_Workspace\rs08sample\Project_Settings\Startup_Code\start08.c:46 0x00003072
C:\CW10_Workspace\rs08sample\RS08\rs08sample.abs (9/3/09 1:50 PM)

Figure 11.48 Debugger Configuration Settings Dialog

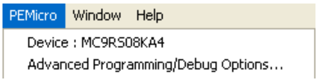

Properties for RS08_KA4_MC9RS08KA4_PnE Cyclone PRO TCP-IP
type filter text
System
System
Edit...
Connection
System
Connection port and Interface Type
Refresh
Compatible Hardware
Socket Programming Options
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
OK
Cancel

RSE system type: Hardware or Simulator
Parent profile: Takao
Connection name: RS08_KA4_MC9RS08KA4_PnE Cyclone PRO TCP-IP
Description: Generated from pre-RSE Launch Configuration
System type: MC9RS08KA4
Connection type: P&E RS08 Multilink\Cyclone Pro
☐ Enable logging
Interface: Cyclone PRO - Ethernet Port
Port: 10.1.0.0 : RS08_KA8
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0
☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

Active Mode Menu Options

When the microprocessor is connected, the active mode menu shows the name of the microprocessor and gives you the access to the Advanced Programming/Debug Options. When the microprocessor is not connected, the menu is not available.

Figure 11.49 Additional Connection Menu Options

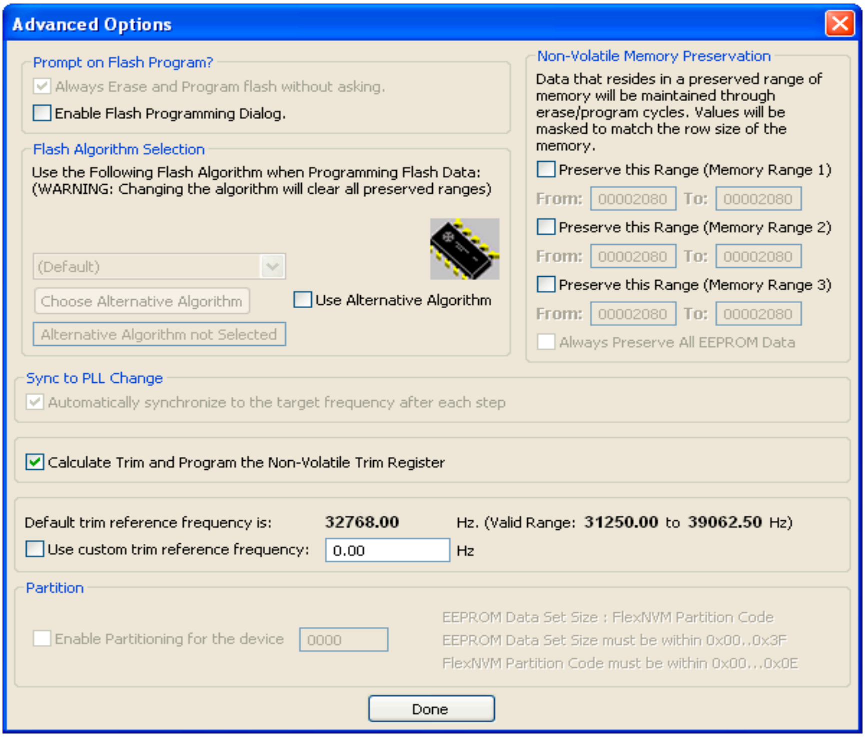


Device : MC9RS08KA4
Advanced Programming/Debug Options...

Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the Flash programming procedure.

Figure 11.50 Advanced Options Dialog Box



Prompt on Flash Program?
☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)
(Default)
Choose Alternative Algorithm
☐ Use Alternative Algorithm
Alternative Algorithm not Selected

Non-Volatile Memory Preservation
Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.
☐ Preserve this Range (Memory Range 1)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 2)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 3)
From: 00002080 To: 00002080
☐ Always Preserve All EEPROM Data

Sync to PLL Change
☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: **32768.00** Hz. (Valid Range: **31250.00** to **39062.50** Hz)
☐ Use custom trim reference frequency: 0.00 Hz

Partition
☐ Enable Partitioning for the device 0000
EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Enable Flash Programming Dialog

Setting the Enable Flash Programming dialog box lets you view the steps taken by the Flash Programmer.

Trim Options

The Calculate Trim and Program the Non-Volatile Trim Register checkbox enables automatic calculation and programming of the trim value to a designated Non-Volatile memory location.

Non-Volatile Memory Preservation

You have the option of preserving up to three independent ranges of non-volatile memory (on devices with EEPROM, the entire EEPROM array may optionally be preserved as well). Ranges that are designated as “preserved” are read before an erase and re-programmed immediately afterwards, thereby preserving the data in these ranges. Any attempts to program data into a preserved range is ignored. When entering an address into the preserved range field (hexadecimal input is expected), the values are masked according to the row size of the device. This ensures that the reprogramming of preserved data does not cause any conditions that disturb programming.

Sync to PLL Change Checkbox

The debugger requires the Sync to PLL Change to synchronize the software/hardware connection with the microprocessor during the Flash erasing/programming procedure.

Trim Control

The Use custom trim reference frequency option lets you select a custom trim value for the target device (valid only for devices with an Internal Clock). You can input any value within the valid internal clock frequency range; the allowable trim value is limited only by the device itself.

NOTE The valid internal clock frequency range and the default trim value for the currently selected device/algorithm are displayed as well.

NOTE For more information about the specific functionality of the internal clock source, see the *Freescale Data Sheet* for your specific device.

Alternative Algorithm Functionality

Once you create a project for a specific HCS08/RS08/CFV1 microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in the `<CW_Install>/MCU/bin/plugins/support/HC08/gdi/P&E` directory.

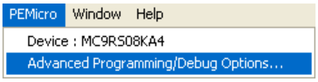
However, the default algorithm may be overridden via the Alternative Algorithm function, located in the Advanced Programming/Debug Options menu. You can use this feature to select a custom programming algorithm, or simply select another one of P&E's many programming algorithms for use with a specific project.

CAUTION Selecting the wrong programming algorithm may damage their device, lead to under/over programming situations, or simply not program portions of the project file. You are recommended to use the default algorithm unless there is a compelling reason to do otherwise.

Use these steps to override the default algorithm:

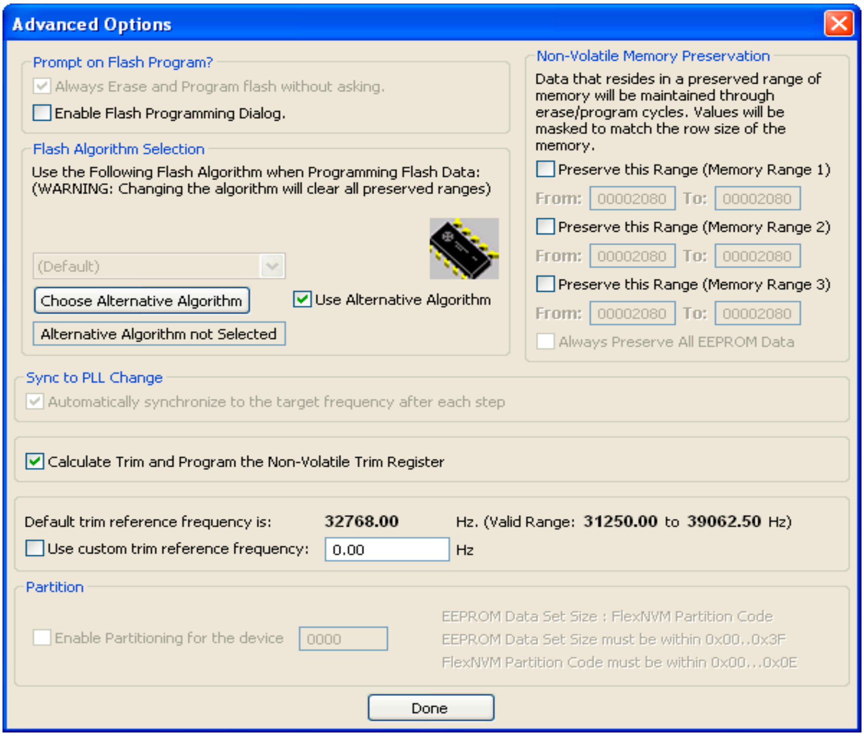
1. Select the **Advanced Programming/Debug Options** selection from the **PEMicro** menu.

Figure 11.51 Advanced Programming/Debug Options Menu Selection



2. Check the Use Alternative Algorithm checkbox.

Figure 11.52 Advanced Options - Alternative Algorithm Checkbox



Advanced Options

Prompt on Flash Program?

☑ Always Erase and Program flash without asking.

☐ Enable Flash Programming Dialog.

Flash Algorithm Selection

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

☑ Use Alternative Algorithm

Alternative Algorithm not Selected

Non-Volatile Memory Preservation

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)

From: 00002080 To: 00002080

☐ Preserve this Range (Memory Range 2)

From: 00002080 To: 00002080

☐ Preserve this Range (Memory Range 3)

From: 00002080 To: 00002080

☐ Always Preserve All EEPROM Data

Sync to PLL Change

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 32768.00 Hz. (Valid Range: 31250.00 to 39062.50 Hz)

☐ Use custom trim reference frequency: 0.00 Hz

Partition

☐ Enable Partitioning for the device 0000

EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

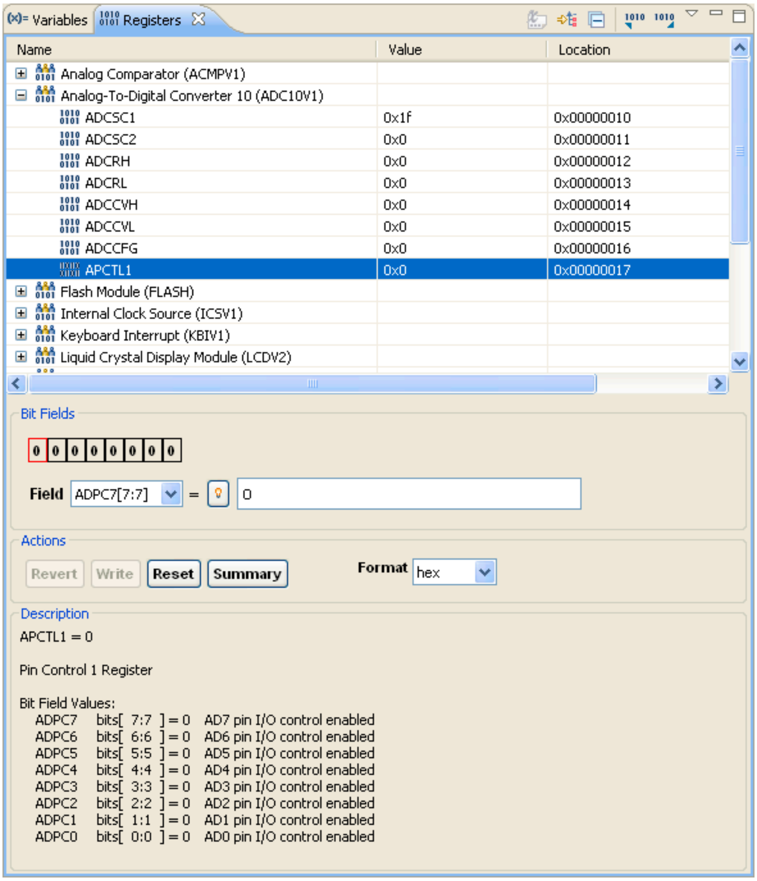
3. Click the **Choose Alternative Algorithm** button, which lets you browse for an alternative algorithm.
4. Once you select the alternative algorithm, the name of the algorithm along with its full path appears in the text field below the **Choose Alternative Algorithm** button.

At this point, the current project performs all future Flash programming operations using the alternative algorithm. You may revert to the default algorithm at any time by clearing the Use Alternative Algorithm checkbox.

View Register Files Option

The Register Files tab in the debugger gives you the option of viewing and editing the register files. If register files are available for the device that you have chosen, the Registers tab in the debugger (see Figure 11.53) is populated.

Figure 11.53 Debug Register File Tab



Name	Value	Location
⊞ Analog Comparator (ACMPV1)		
⊟ Analog-To-Digital Converter 10 (ADC10V1)		
ADCSC1	0x1f	0x00000010
ADCSC2	0x0	0x00000011
ADCRH	0x0	0x00000012
ADCRL	0x0	0x00000013
ADCCVH	0x0	0x00000014
ADCCVL	0x0	0x00000015
ADCCFG	0x0	0x00000016
APCTL1	0x0	0x00000017
⊞ Flash Module (FLASH)		
⊞ Internal Clock Source (ICSV1)		
⊞ Keyboard Interrupt (KBIV1)		
⊞ Liquid Crystal Display Module (LCDV2)		

0	0	0	0	0	0	0	0

Field ADPC7[7:7] = 0

Format hex

```
APCTL1 = 0

Pin Control 1 Register

Bit Field Values:
   ADPC7   bits[  7:7  ] = 0   AD7 pin I/O control enabled
   ADPC6   bits[  6:6  ] = 0   AD6 pin I/O control enabled
   ADPC5   bits[  5:5  ] = 0   AD5 pin I/O control enabled
   ADPC4   bits[  4:4  ] = 0   AD4 pin I/O control enabled
   ADPC3   bits[  3:3  ] = 0   AD3 pin I/O control enabled
   ADPC2   bits[  2:2  ] = 0   AD2 pin I/O control enabled
   ADPC1   bits[  1:1  ] = 0   AD1 pin I/O control enabled
   ADPC0   bits[  0:0  ] = 0   AD0 pin I/O control enabled
```

To view the Register Files of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window
2. Select the **Registers** tab on the right side of the debugging window, or select the **Window > Show View -> Registers** to open the **Register** window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the **Registers** tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit

field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The Actions box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The Description box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

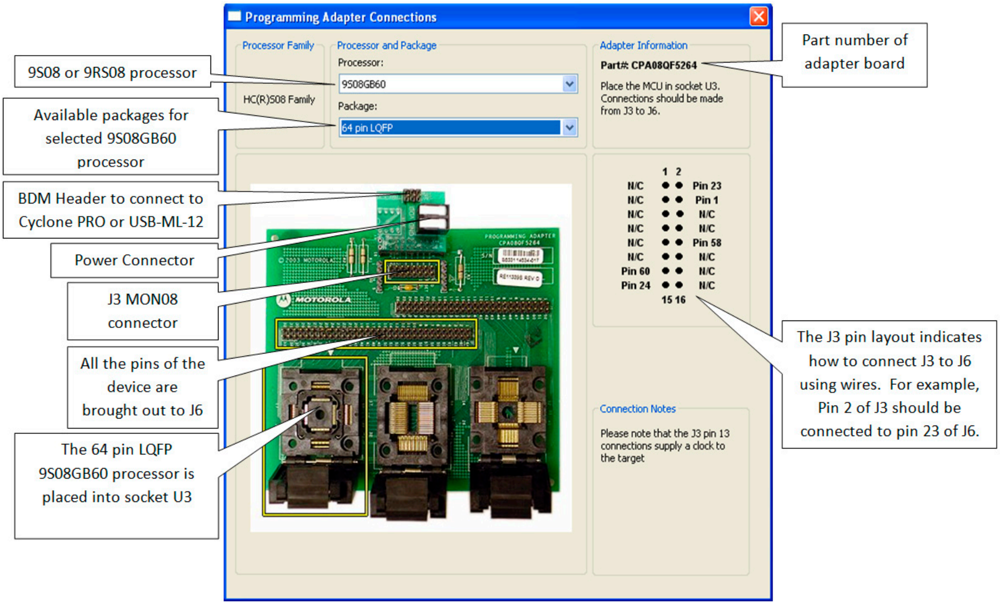
You can modify each bit by selecting it in the drop-down box under the Bit field, or by clicking the bit on the Bit field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

Socket Programming Options Button

The Programming Adapter Connections dialog assistant is designed to facilitate the use of an extensive set of Freescale programming socket adapters. This dialog can be used to get a graphical representation of the signals that must be connected from the BDM header to the pins of the microprocessor. Making these connections lets you establish communication with a given device via a hardware debug interface.

The Socket Programming Options button in the BDM Launch Configuration dialog box (see Figure 11.45) takes you to the Programming Adapter Connections dialog box (see Figure 11.54), where you can look up pin connection settings for the selected package type of the target processor. Only available package types for each target processor are listed in the Package drop-down menu. Once you have selected a package type, the Adapter Information section provides the part number of the adapter board, the socket number where the processor should be placed, and a pair of header numbers that indicate which connections should be made between them. Immediately below the Adapter Information section you will find a pin layout that displays the required connections between the aforementioned pair of headers.

Figure 11.54 Programming Adapter Connections Dialog Box



Programming Adapter Connections
Processor Family
HC(R)S08 Family
Processor and Package
Adapter Information
Connection Notes
1 2
N/C ● ● Pin 23
N/C ● ● Pin 1
N/C ● ● N/C
N/C ● ● N/C
N/C ● ● Pin 58
N/C ● ● N/C
Pin 60 ● ● N/C
Pin 24 ● ● N/C
15 16

Processor: 9S08GB60

Package: 64 pin LQFP

Part#: CPA08QF5264

Place the MCU in socket U3. Connections should be made from J3 to J6.

Please note that the J3 pin 13 connections supply a clock to the target

9S08 or 9RS08 processor

Available packages for selected 9S08GB60 processor

BDM Header to connect to Cyclone PRO or USB-ML-12

Power Connector

J3 MON08 connector

All the pins of the device are brought out to J6

The 64 pin LQFP 9S08GB60 processor is placed into socket U3

Part number of adapter board

The J3 pin layout indicates how to connect J3 to J6 using wires. For example, Pin 2 of J3 should be connected to pin 23 of J6.

P&E RS08 Multilink/Cyclone PRO/OSBDM Connection-Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB BDM Multilink or USB Multilink Universal
- P&E Cyclone PRO Serial
- P&E Cyclone PRO USB
- P&E Cyclone PRO Ethernet
- P&E Open Source BDM

P&E USB BDM Multilink or USB Multilink Universal

The P&E USB BDM Multilink or USB Multilink Universal Connection setting permits a connection to USB BDM Multilink devices. P&E USB BDM Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources. The Multilink connection option can also be used to connect a USB Multilink Universal to your interface.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select P&E USB BDM Multilink as the debugger connection:

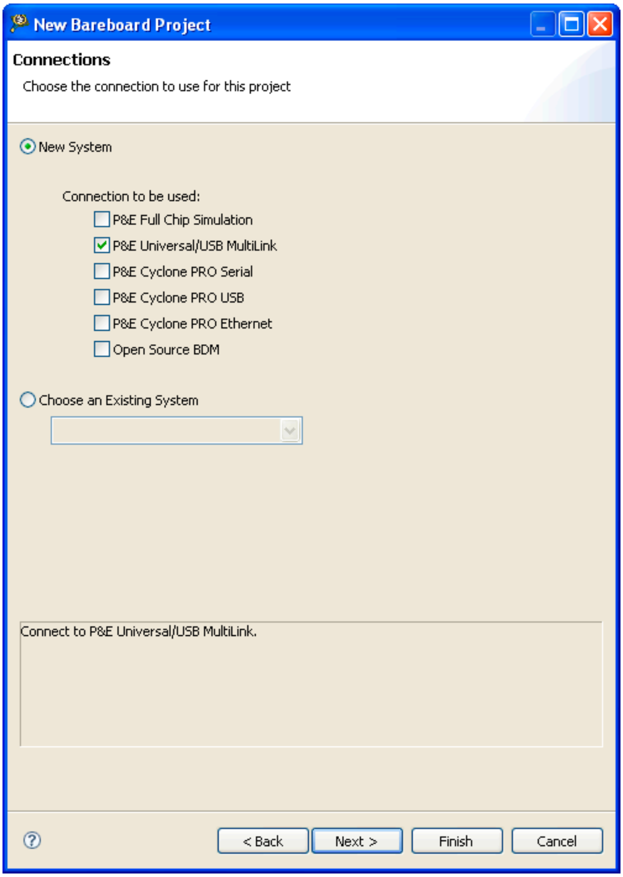
1. Select **Project > Change Device/Connection** from the IDE menu bar. The **Device/ Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse t**o specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**. The **Devices** page appears.
4. Expand the **RS08** tree control and select the derivative or board you would like to use. For example, select **RS08 > RS08KA Family > RS08KA2**.
5. Click **Next**.

   The **Connections** page appears.
6. Check the **P&E Universal/USB Multilink** checkbox. Refer Figure Figure 11.55.

Figure 11.55 RS08 P&E USB BDM Multilink Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☑ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

7. Click **Finish**.

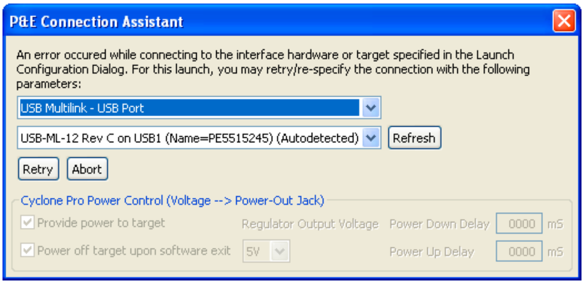
The wizard creates a project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E USB BDM Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down menu and click **Refresh**. Refer Figure 11.56.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 11.56 RS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO Serial

The P&E Cyclone PRO Serial Connection setting permits a connection to Cyclone PRO Serial devices. P&E Cyclone PRO Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select P&E Cyclone PRO Serial as the debugger connection:

1. Select Project > Change Device/Connection from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location checkbox** is checked.

3. Click **Next**.

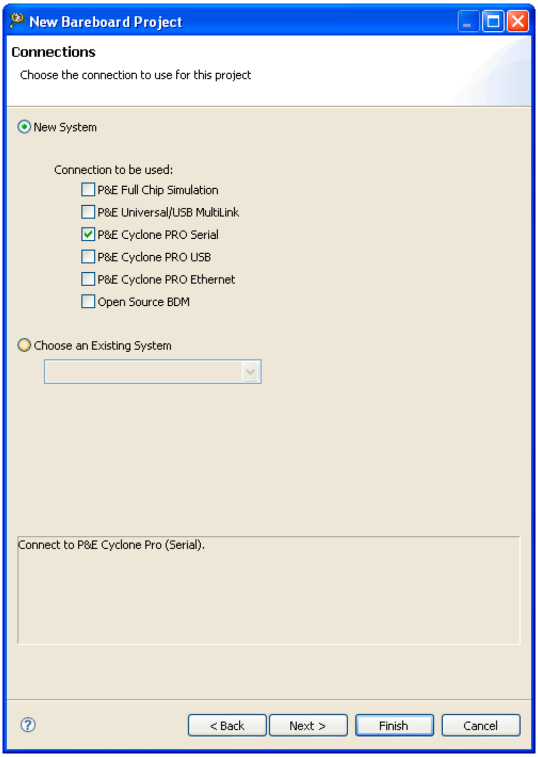
   The **Devices** page appears.

4. Expand the **RS08** tree control and select the derivative or board you would like to use. For example, select **RS08 > RS08KA Family > RS08KA2**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO Serial** checkbox. Refer Figure 11.57.

Figure 11.57 RS08 P&E Cyclone PRO Serial Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☐ P&E Universal/USB MultiLink
- ☑ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (Serial).

7. Click **Finish**.

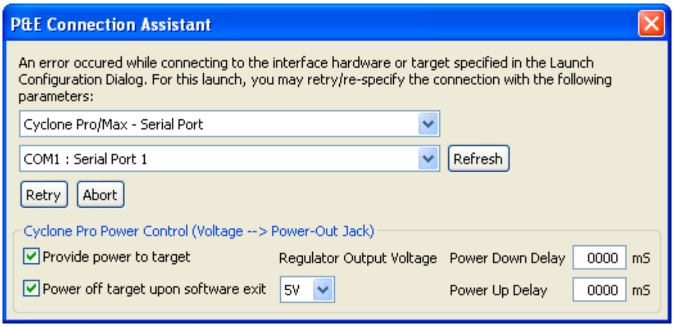
The wizard creates a project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down menu and click **Refresh**. Refer Figure 11.58.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 11.1 for more details of each setting.
4. Click the **Retry** button.

Figure 11.58 RS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port

COM1 : Serial Port 1

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO USB

The P&E Cyclone PRO USB Connection setting permits a connection to Cyclone PRO USB devices. P&E Cyclone PRO USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone PRO USB** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location checkbox** is checked.

3. Click **Next**.

   The **Devices** page appears.

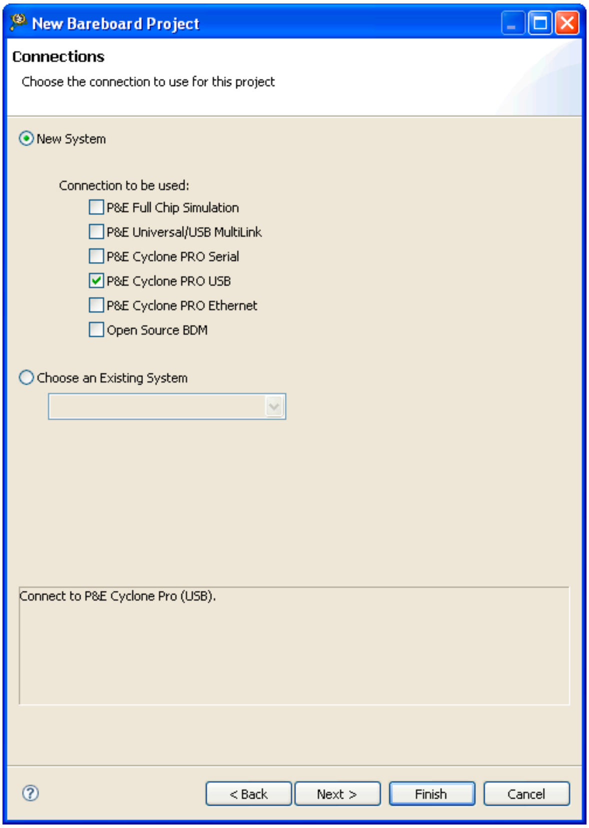
4. Expand the **RS08** tree control and select the derivative or board you would like to use. For example, select **RS08 > RS08KA Family > RS08KA2**.

5. Click **Next.**

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO USB** checkbox. Refer Figure 11.59.

Figure 11.59 RS08 P&E Cyclone PRO USB Selected


New Bareboard Project
Connections
Choose the connection to use for this project

● New System

Connection to be used:
- ☐ P&E Full Chip Simulation
- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☑ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (USB).

< Back
Next >
Finish
Cancel

7. Click **Finish**.

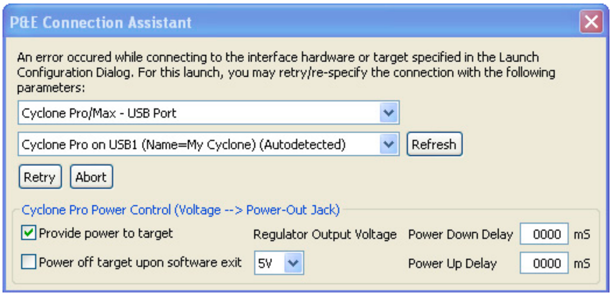
The wizard creates a project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO USB** as your debugger connection:

1. Select **Cyclone Pro/Max – USB Port** from the first drop-down menu and click **Refresh**. Refer Figure 11.60.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 11.1 for more details of each setting.
4. Click the **Retry** button.

Figure 11.60 RS08 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Pro on USB1 (Name=My Cyclone) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target
☐ Power off target upon software exit
Regulator Output Voltage: 5V
Power Down Delay: 0000 mS
Power Up Delay: 0000 mS

P&E Cyclone PRO Ethernet

The P&E Cyclone PRO Ethernet Connection setting permits a connection to Cyclone PRO Ethernet devices. P&E Cyclone PRO Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select P&E Cyclone PRO Ethernet as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location checkbox i**s checked.

3. Click **Next**.

   The **Devices** page appears.

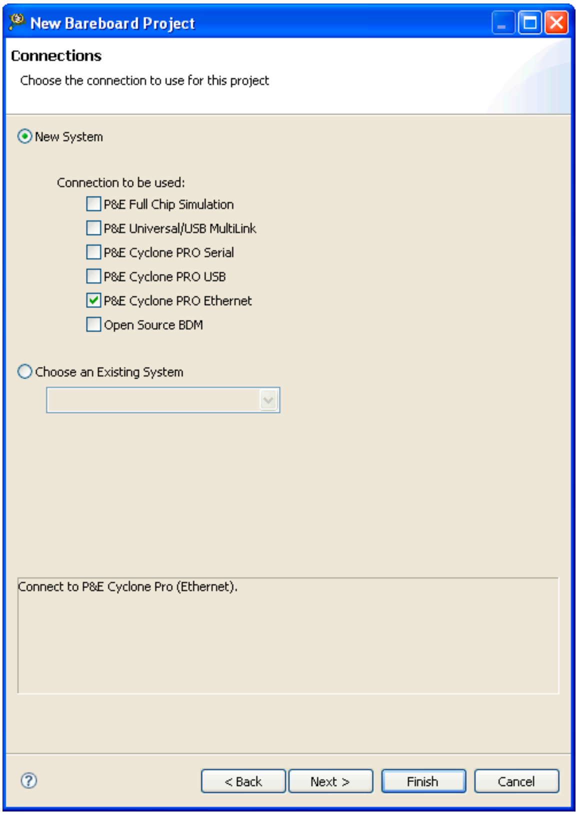
4. Expand the RS08 tree control and select the derivative or board you would like to use. For example, select **RS08 > RS08KA Family > RS08KA2.**

5. Click **Next**.

   The **Connection**s page appears.

6. Check the **P&E Cyclone PRO Ethernet** checkbox. Refer Figure 11.61.

Figure 11.61 RS08 P&E Cyclone PRO Ethernet Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Full Chip Simulation
- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☑ P&E Cyclone PRO Ethernet
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (Ethernet).

7. Click **Finish**.

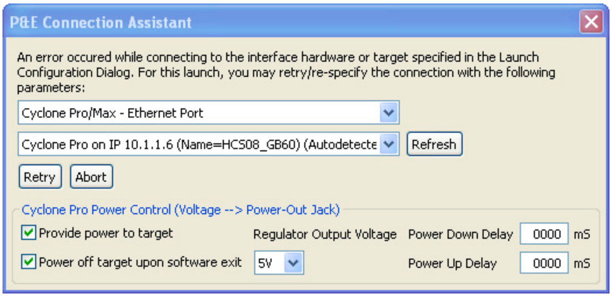
The wizard creates a project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down menu and click **Refresh**. Refer Figure 11.62.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 11.1 for more details of each setting.
4. Click the **Retry** button

Figure 11.62 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Pro on IP 10.1.1.6 (Name=HCS08_GB60) (Autodetecte

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage 5V

Power Down Delay 0000 mS

Power Up Delay 0000 mS

P&E Open Source BDM

Freescale supplies certain development boards with an integrated debug circuit based on Open Source BDM. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Pro or USB Multilink. The development board also derives its power from the USB Bus.

The Open Source BDM circuit design (OSBDM-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source BDM" connection. P&E has integrated the Open Source BDM support into the same connection that supports both the USB Multilink and the Cyclone Pro. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSBDM (albeit at a lower data rate).

The P&E Open Source BDM Connection setting permits a connection to Open Source BDM devices. Open Source BDM mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **Open Source BDM** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse t**o specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

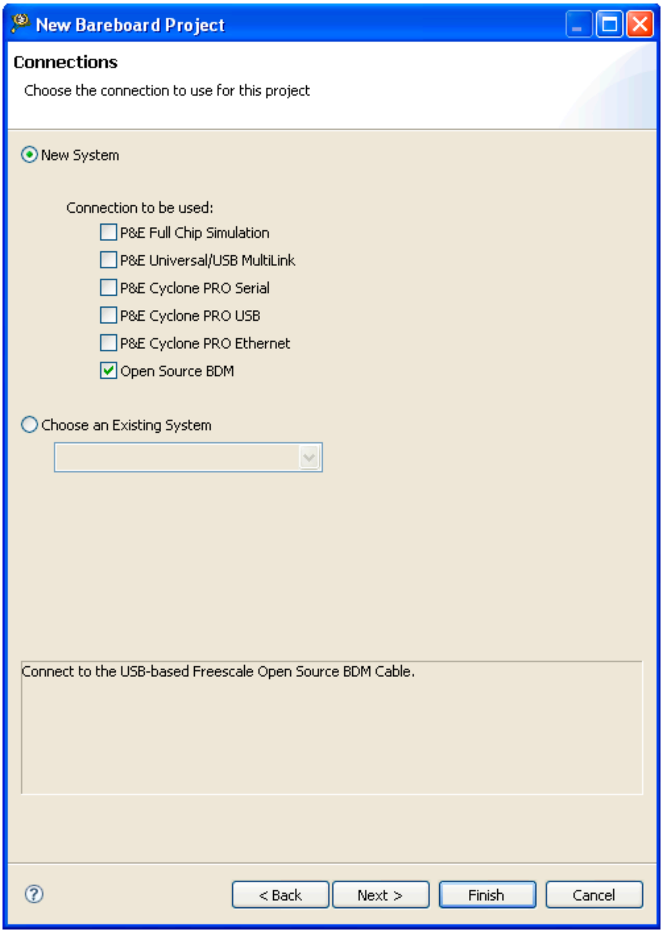
   The **Devices** page appears.

4. Expand the **RS08** tree control and select the derivative or board you would like to use. For example, select **RS08 > RS08KA Family > RS08KA2**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the P**&E Open Source BDM** checkbox. Refer to Figure 11.63.

Figure 11.63 RS08 P&E Open Source BDM Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:
- ☐ P&E Full Chip Simulation
- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☑ Open Source BDM

○ Choose an Existing System

Connect to the USB-based Freescale Open Source BDM Cable.

7. Click Finish.

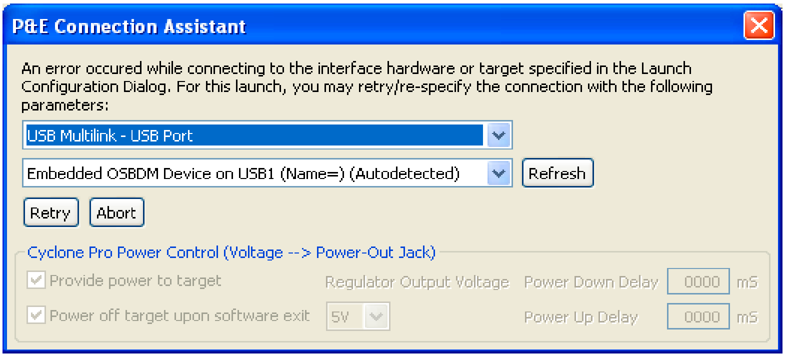
The wizard creates a project for the RS08 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Open Source BDM** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down menu and click **Refresh**. Refer to Figure 11.64.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 11.64 RS08 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

Embedded OSBDM Device on USB1 (Name=) (Autodetected)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

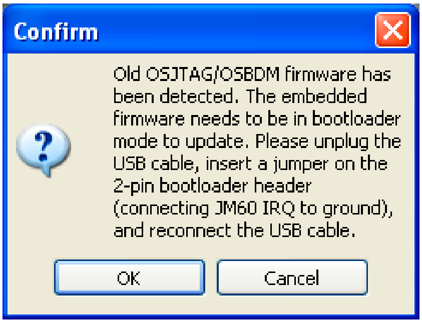
Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

OSBDM Firmware Update

All Codewarrior IDE's version 10.1 and higher have an automatic firmware update mechanism for built-in OSBDM hardware interfaces. Whenever an OSBDM-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSBDM firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

Figure 11.65 Old OSBDM Firmware Detected

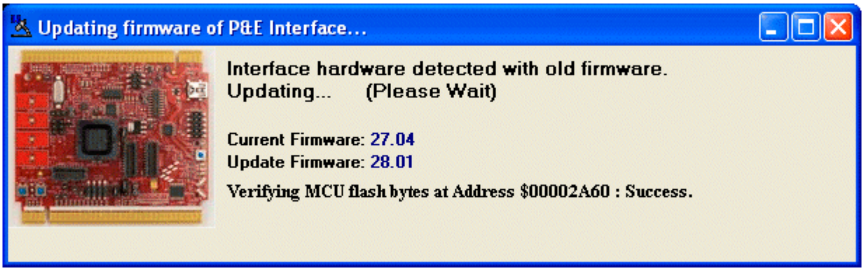


Confirm

Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSBDM device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSBDM-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSBDM device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSBDM device and click on the OK button. If done correctly, the automatic firmware update will occur.

Figure 11.66 OSBDM Firmware Updating

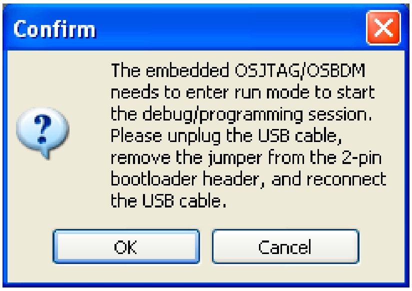


Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01
Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSBDM device must exit Bootloader mode and enter into Run mode.

Figure 11.67 Start OSBDM Run Mode



The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSBDM device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click on OK and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSBDM firmware. If for any reason you experience difficulty performing OSBDM firmware update, please visit www.pemicro.com/osbdm and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

12

Connections — ColdFire V1

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the **ColdFire V1** full chip simulator or the target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the **Launch Configuration Settings** panel. **Launch Configuration Settings** panel can be accessed by clicking on the **Edit** button located within the **Main** tab of the **Debug Configurations** dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with simulation platforms and hardware devices that are part of the ColdFire V1 device family.

The topics in this chapter are:

- Changing Connection in IDE
- P&E USB BDM Multilink/Cyclone Pro/OSBDM
- CCS
- CCS

Changing Connection in IDE

To change connection in the IDE, perform these steps.

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the tree control and select the derivative or board you would like to use.
5. Click **Next**.

   The **Connections** page appears.

6. Select the desired connection.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

P&E USB BDM Multilink/Cyclone Pro/ OSBDM

This section describes the CFV1 P&E Connection options. The CFV1 P&E Connection setting permits a connection to CFV1 Freescale devices via P&E Multilink/Cyclone Pro/ OSBDM hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

This topic describes all P&E USB BDM Multilink/Cyclone Pro/OSBDM connection options, which are common to all P&E USB BDM Multilink or USB Multilink Universal, P&E Cyclone PRO Serial, P&E Cyclone PRO USB, P&E Cyclone PRO Ethernet, and P&E Open Source BDM connections.

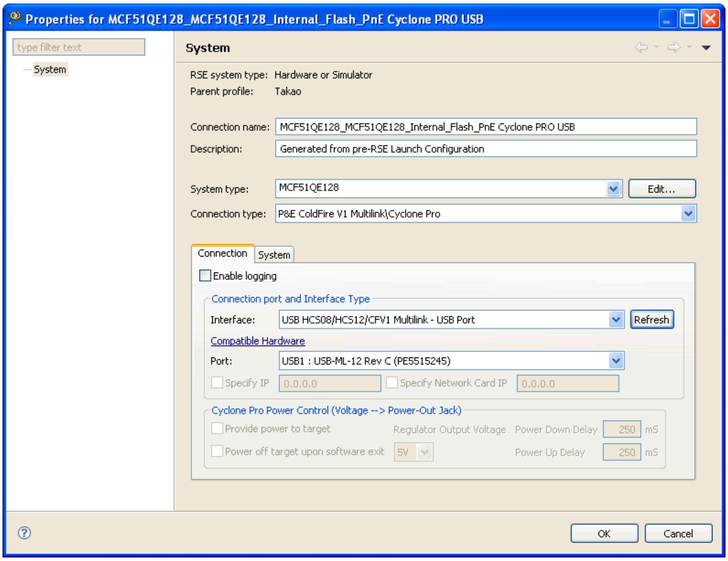
The options include:

- Changing P&E Connection Settings
- Connection Assistant
- Active Mode Menu Options
- Advanced Programming/Debug Options
- View Register Files Options
- P&E USB BDM Multilink/Cyclone PRO/OSBDM Connection-Specific Options

Changing P&E Connection Settings

All connection settings for P&E hardware interfaces are configured in the Launch Configurations dialog box.

Figure 12.1 GDI — P&E USB BDM Multilink/Cyclone PRO/OSBDM Connection



Properties for MCF51QE128_MCF51QE128_Internal_Flash_PnE Cyclone PRO USB

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF51QE128_MCF51QE128_Internal_Flash_PnE Cyclone PRO USB
Description: Generated from pre-RSE Launch Configuration

System type: MCF51QE128
Connection type: P&E ColdFire V1 Multilink\Cyclone Pro

☐ Enable logging

Connection port and Interface Type
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port
Port: USB1 : USB-ML-12 Rev C (PE5515245)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☐ Provide power to target
☐ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

Table 12.1 describes the options for this view.

Table 12.1 Connection Parameter Options for P&E USB BDM Multilink/Cyclone Pro/ OSBDM

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB HCS08\HCS12\CFV1 Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone PRO - Serial Port • Cyclone PRO - USB Port • Cyclone PRO - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project.
Refresh	Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE:** If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help.
Socket Programming Options	The Socket Programming Options button brings up a dialog that provides you a graphical representation of the signals that must be connected from the BDM header to the pins of the microprocessor, in order to use Freescale socket adapters.

Table 12.1 Connection Parameter Options for P&E USB BDM Multilink/Cyclone Pro/OSBDM (*continued*)

Option	Description
(Cyclone Ethernet only) Specify IP	Use this option to specify the IP address of a Cyclone outside of the local network. Click on the checkbox to enable the textbox. This will also disable the port dropdown box. Currently supports IPv4 only.
(Cyclone Ethernet only) Specify Network Card IP	Use this option to specify the local network card IP address if there are multiple cards on your computer. Click on the checkbox to enable the textbox. Currently supports IPv4 only.
(Cyclone PRO only) Provide power to target	This option determines whether the Cyclone PRO (circuitry) provides power to the target hardware via the probe. Check this option to have the Cyclone PRO (circuitry) supply power to the hardware target Uncheck this option to not provide power.
(Cyclone PRO only) Power off target upon software exit	This option determines whether Cyclone PRO hardware interface provides power to the target hardware via VDD of the BDM cable. Check this option to turn off the power when the program terminates. Uncheck this option to leave the hardware target powered continuously.
(Cyclone PRO only) Regulator Output Voltage	This option adjusts the output voltage that powers the hardware target. Select a voltage value from this option’s list box. **NOTE:** An improper voltage setting can damage the board.

Table 12.1 Connection Parameter Options for P&E USB BDM Multilink/Cyclone Pro/OSBDM (*continued*)

Option	Description
(Cyclone PRO only) Power down delay	This option specifies amount of time for which the target will be turned off during a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option’s text box.
(Cyclone PRO only) Power up delay	This option specifies amount of time for which the target will remain powered prior to a RESET power cycling sequence. Enter the delay interval (in milliseconds) in this option’s text box.

WARNING! An improper voltage setting can damage the board.

To change P&E Connections settings, perform these steps.

1. In the **CodeWarrior Projects** view, select the project for which you want to change the P&E Connections settings.

NOTE It is assumed that you have created a project and built it.

2. Select **Run > Debug Configurations** from the main menu bar of the IDE.

   The **Debug Configurations** dialog box appears.

3. Expand the **CodeWarrior Download** tree control in the left pane and select the launch configuration you want to debug.
4. Click the **Main** tab.

   The **Main** page appears in the area beneath the tabs.

5. Select a system within **Remote System** which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the Freescale Eclipse Extensions Guide. Once a remote system is selected, click the **Edit** button. The **Launch Configuration Settings Panel** will appear.
6. Ensure that the **System Type** is the correct MCU you want to debug. Use the drop-down box or the **Edit** button to change this option.
7. In the **Connection Type** drop-down box, select P&E ColdFire V1 Multilink\Cyclone Pro. The P&E connections settings will appear below.

8. Click **Refresh** to scan valid interface and port.

   Valid interfaces and ports appear in the **Interface** and **Port** drop-down lists in the **Connection** port and **Interface Type** group.

9. Select a supported interface from the **Interface** drop-down list.
10. Select a supported port from the **Port** drop-down list.

NOTE The port displayed may vary depending on the interface. For example, if you select interface as Cyclone PRO - Serial Port, the available port option is COM1 : Serial Port 1.

11. Specify settings in the **Cyclone Pro Power Control (Voltage --> Power -Out Jack)** group.

NOTE This group will be enabled for Cyclone PRO interface only.

- Check the **Provide power to target** checkbox to have the Cyclone PRO (circuitry) provide power to the target else clear the checkbox if you do not want to provide power to the target.
- Check the **Power off target upon software exit** checkbox to turn off the power when the program terminate else clear the checkbox to leave the hardware target powered continuously.
- Select a voltage value from the **Regulator Output Voltage** drop-down list. This adjusts the output voltage that powers the hardware target.

WARNING! An improper voltage setting can damage the board.

- Enter the delay interval (in milliseconds) in the **Power Down Delay** text box. This option specifies the time interval to wait before shutting off the power to the hardware target. Cyclone Pro powers down the device once the debug session is over, or while executing a power cycling sequence after beginning a new debug session.
- Enter the delay interval (in milliseconds) in the **Power Up Delay** text box. This option specifies the time interval to wait before turning on the power to the hardware target. If the Cyclone Pro power to target feature is enabled, Cyclone Pro will power up the device while executing a power cycling sequence at the beginning of every debug session.

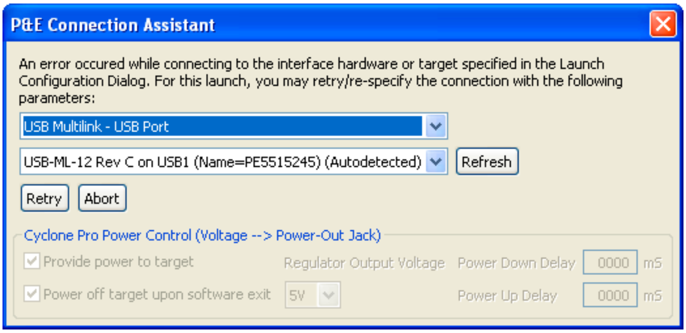
12. Click **OK** to save changes to the P&E Connections settings. The Launch Configuration Settings Panel dialog box will close.
13. Click **Close** button to close the Debug Configuration dialog box.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug but the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the P&E USB BDM Multilink/Cyclone PRO/OSBDM as your debugger connection:

1. Select the P&E device that you are using from the first drop-down menu and click **Refresh**. Refer Figure 12.2.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings (Cyclone Pro only). Refer to Table 12.1 for more details of each setting.
4. Click the **Retry** button

Figure 12.2 CFV1 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

Launch Configuration Settings

To set the launch configurations for the debugger:

1. Find the debugger icon and click on the drop-down arrow to bring up the debugger menu. Refer Figure 12.2.
2. Select Debug Configurations
3. In the left column, select the project download type for which you would like to set the launch configurations. Refer Figure 12.3
4. In the right column, click on the **Debugger** tab.
5. Set your configurations and click **Debug** to start the debugger.

Figure 12.3 Debugger Drop-down Menu

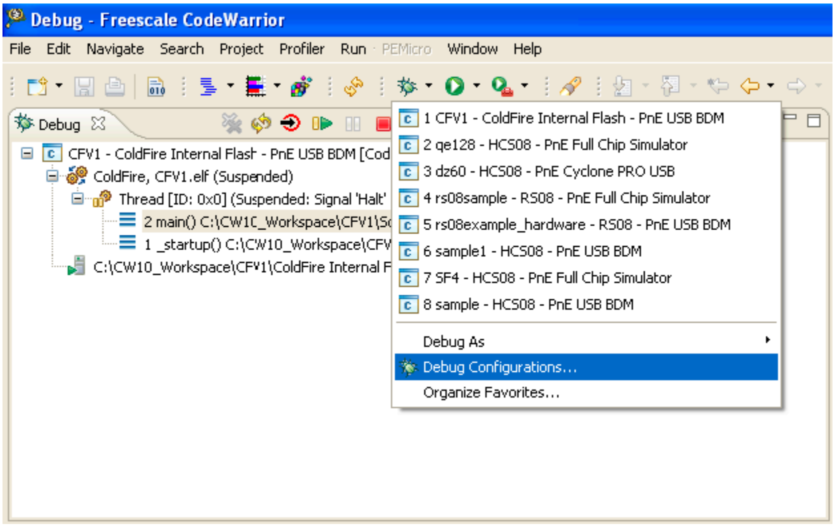
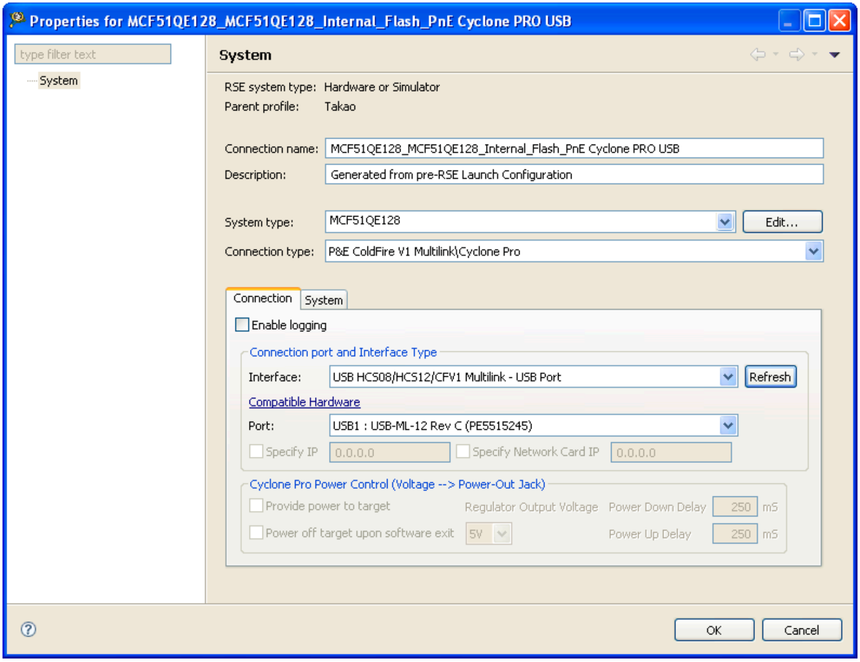

Debug - Freescale CodeWarrior
File Edit Navigate Search Project Profiler Run PEMicro Window Help
Debug
CFV1 - ColdFire Internal Flash - PnE USB BDM [Cod
ColdFire, CFV1.elf (Suspended)
Thread [ID: 0x0] (Suspended: Signal 'Halt'
2 main() C:\CW10_Workspace\CFV1\Sc
1 _startup() C:\CW10_Workspace\CFV
C:\CW10_Workspace\CFV1\ColdFire Internal F
1 CFV1 - ColdFire Internal Flash - PnE USB BDM
2 qe128 - HCS08 - PnE Full Chip Simulator
3 dz60 - HCS08 - PnE Cyclone PRO USB
4 rs08sample - RS08 - PnE Full Chip Simulator
5 rs08example_hardware - RS08 - PnE USB BDM
6 sample1 - HCS08 - PnE USB BDM
7 SF4 - HCS08 - PnE Full Chip Simulator
8 sample - HCS08 - PnE USB BDM
Debug As
Debug Configurations...
Organize Favorites...

Figure 12.4 Debugger Configuration Settings Dialog Box



Properties for MCF51QE128_MCF51QE128_Internal_Flash_PnE Cyclone PRO USB

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF51QE128_MCF51QE128_Internal_Flash_PnE Cyclone PRO USB
Description: Generated from pre-RSE Launch Configuration

System type: MCF51QE128
Connection type: P&E ColdFire V1 Multilink\Cyclone Pro

☐ Enable logging

Connection port and Interface Type
Interface: USB HCS08/HCS12/CFV1 Multilink - USB Port
Port: USB1 : USB-ML-12 Rev C (PE5515245)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

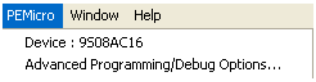
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☐ Provide power to target
☐ Power off target upon software exit
Regulator Output Voltage 5V
Power Down Delay 250 mS
Power Up Delay 250 mS

type filter text
System
Edit...
Connection
System
Refresh
Compatible Hardware
OK
Cancel

Active Mode Menu Options

When the microprocessor is connected, the active mode menu shows the name of the microprocessor and gives you access to the Advanced Programming/Debug Options. When the microprocessor is not connected, the menu is not available.

Figure 12.5 Additional Connection Menu Options

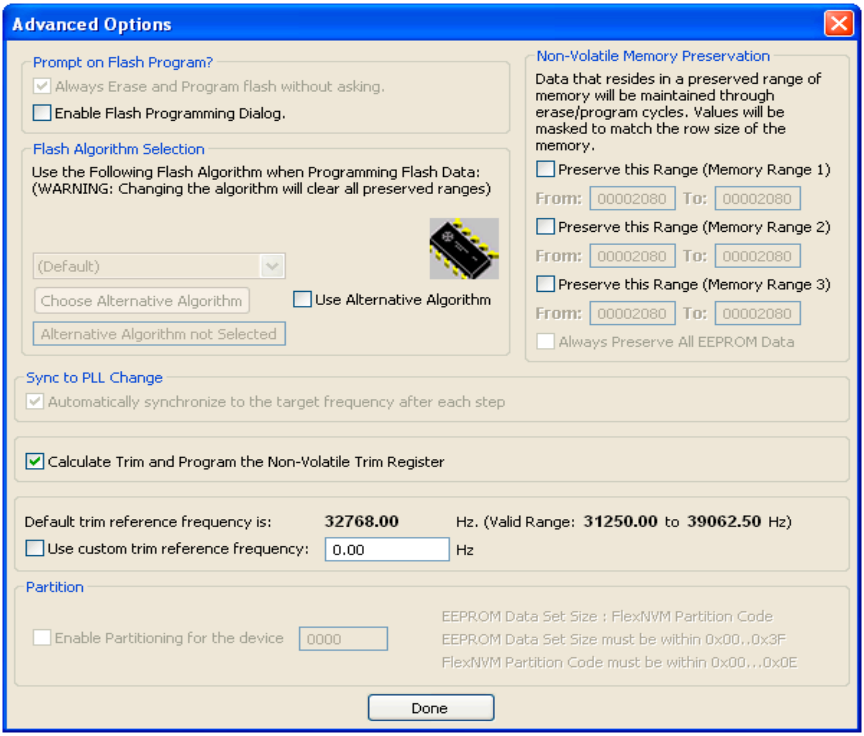


PEMicro
Window
Help
Device : 9S08AC16
Advanced Programming/Debug Options...

Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the Flash programming procedure.

Figure 12.6 Advanced Options Dialog Box



Advanced Options

Prompt on Flash Program?

☑ Always Erase and Program flash without asking.

☐ Enable Flash Programming Dialog.

Flash Algorithm Selection

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

Choose Alternative Algorithm

☐ Use Alternative Algorithm

Alternative Algorithm not Selected

Non-Volatile Memory Preservation

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)

From: 00002080 To: 00002080

☐ Preserve this Range (Memory Range 2)

From: 00002080 To: 00002080

☐ Preserve this Range (Memory Range 3)

From: 00002080 To: 00002080

☐ Always Preserve All EEPROM Data

Sync to PLL Change

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: **32768.00** Hz. (Valid Range: **31250.00** to **39062.50** Hz)

☐ Use custom trim reference frequency: 0.00 Hz

Partition

☐ Enable Partitioning for the device 0000

EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Enable Flash Programming Dialog

Setting the Enable Flash Programming dialog box lets you view the steps taken by the Flash Programmer.

Trim Options

The Calculate Trim and Program the Non-Volatile Trim Register checkbox enables automatic calculation and programming of the trim value to a designated Non-Volatile memory location.

Non-Volatile Memory Preservation

You have the option of preserving up to three independent ranges of non-volatile memory (on devices with EEPROM, the entire EEPROM array may optionally be preserved as well). Ranges that are designated as “preserved” are read before an erase and reprogrammed immediately afterwards, thereby preserving the data in these ranges. Any attempts to program data into a preserved range are ignored. When entering an address into the preserved range field (hexadecimal input is required), the values are masked according to the row size of the device. This ensures that the reprogramming of preserved data does not cause any conditions that disturb programming.

Sync to PLL Change Checkbox

The debugger requires the Sync to PLL Change to synchronize the software/hardware connection with the microprocessor during the Flash erase/program procedure.

Trim Control

The Use custom trim reference frequency option lets you select a custom trim value for the target device (valid only for devices with an Internal Clock). You can input any value within the valid internal clock frequency range; the allowable trim value is limited only by the device itself. Note that the valid internal clock frequency range and the default trim value for the currently selected device/algorithm are displayed as well. For more information about the specific functionality of the internal clock source, refer to the Freescale Data Sheet for your specific device.

Alternative Algorithm Functionality

Once you create a project for a specific HCS08/RS08/CFV1 microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in the `<CW_Install>/MCU/bin/plugins/support/Coldfire/gdi/P&E` directory.

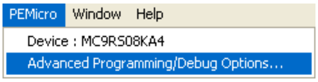
However, the default algorithm may be overridden via the Alternative Algorithm function, located in the Advanced Programming/Debug Options menu. You can use this feature to select a custom programming algorithm, or simply select another one of P&E's many programming algorithms for use with a specific project.

CAUTION Selecting the wrong programming algorithm may damage their device, lead to under/over programming situations, or simply not program portions of the project file. It is therefore recommended using the default algorithm unless there is a compelling reason to do otherwise.

Use these steps to override the default algorithm:

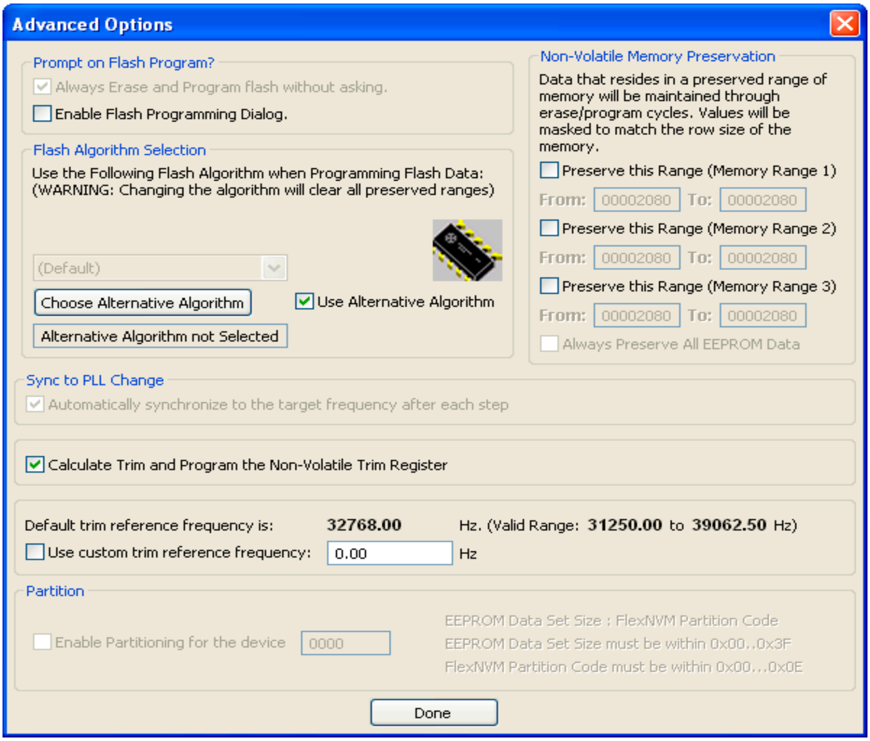
1. Select the Advanced Programming/Debug Options selection from the PEMicro menu.

Figure 12.7 Advanced Programming/Debug Options Menu Selection


PEMicro
Window
Help
Device : MC9RS08KA4
Advanced Programming/Debug Options...

2. Check the Use Alternative Algorithm checkbox.

Figure 12.8 Advanced Options - Alternative Algorithm Checkbox


Advanced Options
Prompt on Flash Program?
Flash Algorithm Selection
Choose Alternative Algorithm
Non-Volatile Memory Preservation
Sync to PLL Change
Partition
Done

☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

☑ Use Alternative Algorithm

Alternative Algorithm not Selected

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 2)
From: 00002080 To: 00002080
☐ Preserve this Range (Memory Range 3)
From: 00002080 To: 00002080
☐ Always Preserve All EEPROM Data

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 32768.00 Hz. (Valid Range: 31250.00 to 39062.50 Hz)
☐ Use custom trim reference frequency: 0.00 Hz

☐ Enable Partitioning for the device 0000
EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

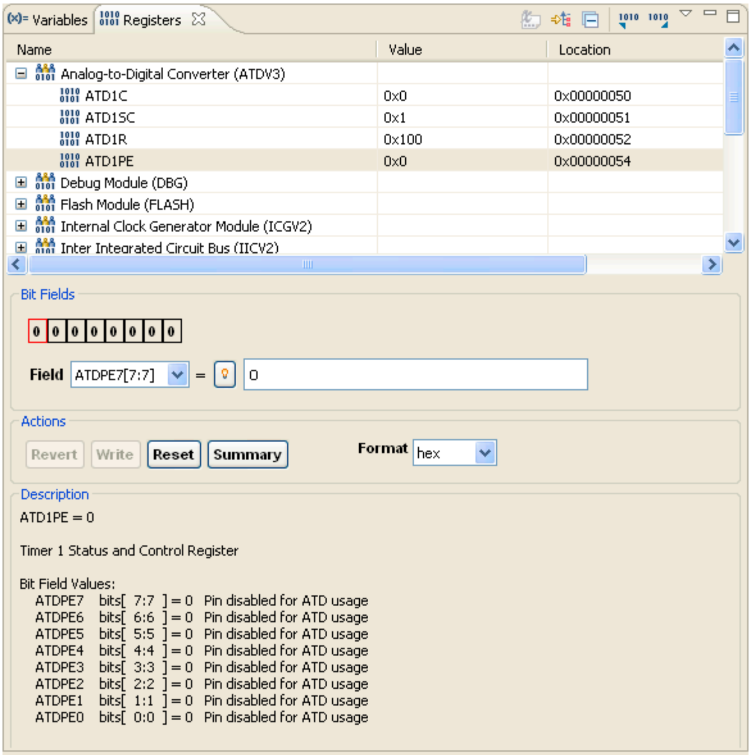
3. Click the **Choose Alternative Algorithm** button, which lets you browse for an alternative algorithm.
4. Once you select the alternative algorithm, the name of the algorithm along with its full path appears in the text field below the **Choose Alternative Algorithm** button.

At this point, the current project performs all future Flash programming operations using the alternative algorithm. You may revert to the default algorithm at any time by clearing the Use Alternative Algorithm checkbox.

View Register Files Options

The Register Files tab in the debugger gives you the option of viewing and editing the register files. If register files are available for the device that you have chosen, the Registers tab in the debugger (see Figure 12.9) is populated.

Figure 12.9 Debug Register File Tab



Name	Value	Location
⊟ Analog-to-Digital Converter (ATDV3)		
ATD1C	0x0	0x00000050
ATD1SC	0x1	0x00000051
ATD1R	0x100	0x00000052
ATD1PE	0x0	0x00000054
⊞ Debug Module (DBG)		
⊞ Flash Module (FLASH)		
⊞ Internal Clock Generator Module (ICGV2)		
⊞ Inter Integrated Circuit Bus (IICV2)		

0 0 0 0 0 0 0 0

Field ATDPE7[7:7] = 0

Format hex

ATD1PE = 0

Timer 1 Status and Control Register

```
Bit Field Values:
   ATDPE7   bits[ 7:7 ] = 0   Pin disabled for ATD usage
   ATDPE6   bits[ 6:6 ] = 0   Pin disabled for ATD usage
   ATDPE5   bits[ 5:5 ] = 0   Pin disabled for ATD usage
   ATDPE4   bits[ 4:4 ] = 0   Pin disabled for ATD usage
   ATDPE3   bits[ 3:3 ] = 0   Pin disabled for ATD usage
   ATDPE2   bits[ 2:2 ] = 0   Pin disabled for ATD usage
   ATDPE1   bits[ 1:1 ] = 0   Pin disabled for ATD usage
   ATDPE0   bits[ 0:0 ] = 0   Pin disabled for ATD usage
```

To view the Register Files of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window
2. Select the “Registers” tab on the right side of the debugging window, or select the Window menu -> Show View -> Registers to open the Register window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the Registers tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The Actions box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The Description box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

You can modify each bit by selecting it in the drop-down box under the **Bit** field, or by clicking the bit on the **Bit** field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

P&E USB BDM Multilink/Cyclone PRO/ OSBDM Connection-Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB BDM Multilink or USB Multilink Universal
- P&E Cyclone PRO Serial
- P&E Cyclone PRO USB
- P&E Cyclone PRO Ethernet
- P&E Open Source BDM

P&E USB BDM Multilink or USB Multilink Universal

The P&E USB BDM Multilink or USB Multilink Universal Connection setting permits a connection to USB BDM Multilink devices. P&E USB BDM Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources. The Multilink connection option can also be used to connect a USB Multilink Universal to your interface.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E USB BDM Multilink** or **USB Multilink Universal** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next.

   The **Devices** page appears.

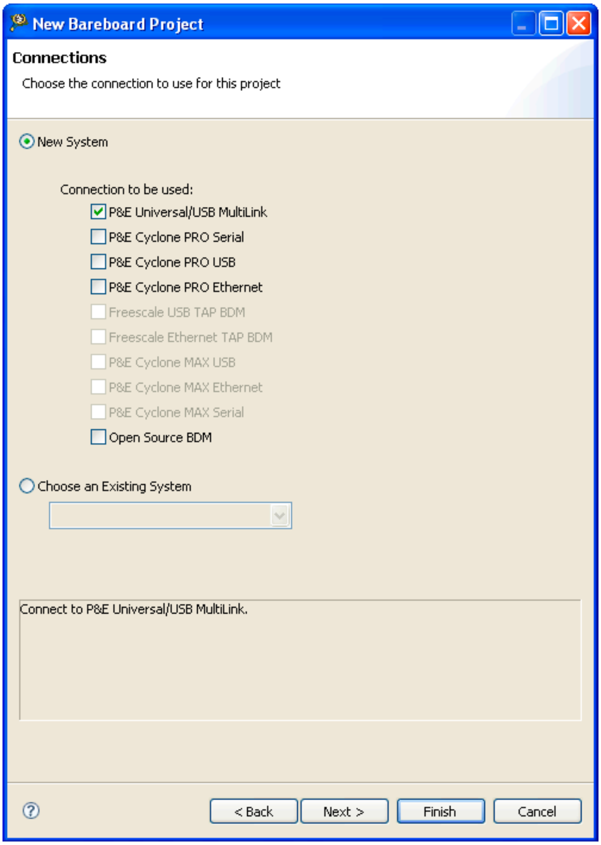
4. Expand the RS08 tree control and select the derivative or board you would like to use. For example, select **ColdFire V1 > MCF51QE Family > MCF51QE128**.

5. Click Next.

   The **Connections** page appears.

6. Check the **P&E Universal/USB Multilink** checkbox. Refer Figure 12.10.

Figure 12.10 ColdFire V1 P&E USB BDM Multilink Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☑ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

7. Click **Finish**.

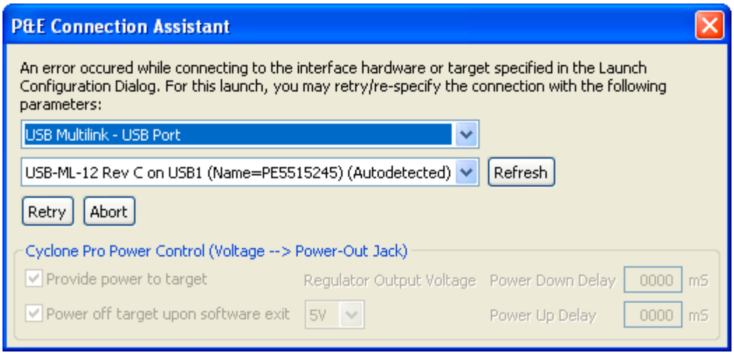
The wizard creates a project for the ColdFire V1 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E USB BDM Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down menu and click **Refresh**. Refer Figure 12.11.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 12.11 ColdFire V1 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-12 Rev C on USB1 (Name=PE5515245) (Autodetected)

☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage: 5V
Power Down Delay: 0000 mS
Power Up Delay: 0000 mS

P&E Cyclone PRO Serial

The P&E Cyclone PRO Serial Connection setting permits a connection to Cyclone PRO Serial devices. P&E Cyclone PRO Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone PRO Serial** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next.

   The **Devices** page appears.

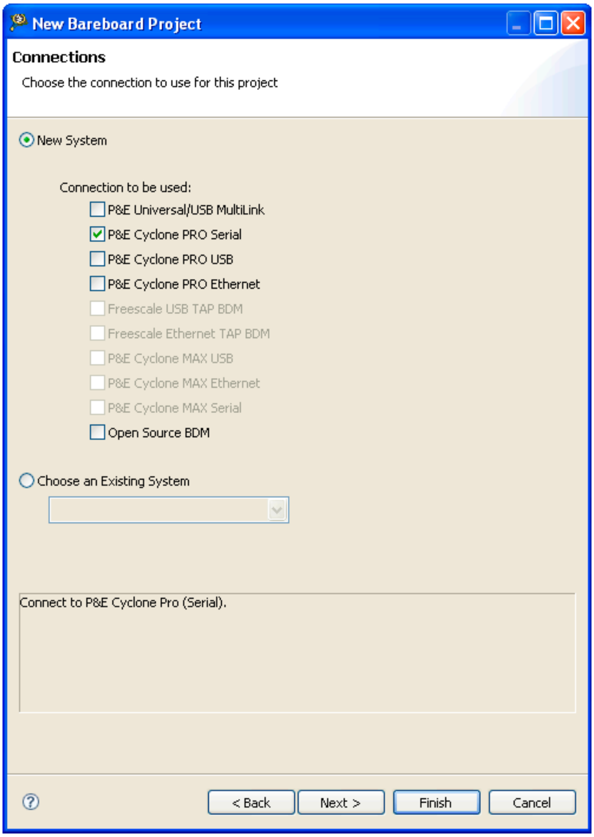
4. Expand the ColdFire V1 tree control and select the derivative or board you would like to use. For example, select ColdFire V1 >MCF51QE Family > MCF51QE128.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO Serial** checkbox. Refer Figure 12.12.

Figure 12.12 ColdFire V1 P&E Cyclone PRO Serial Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Universal/USB MultiLink
- ☑ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (Serial).

7. Click **Finish**.

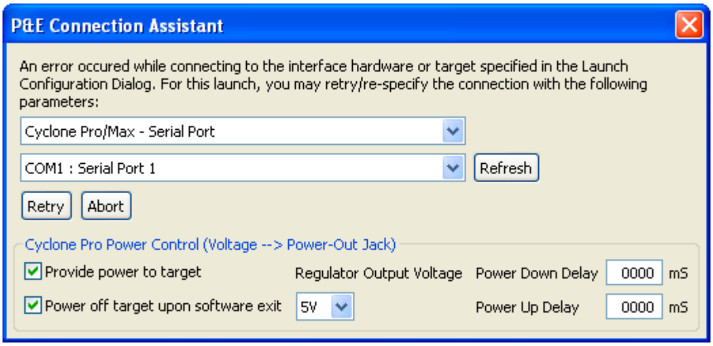
The wizard creates a project for the ColdFire V1 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down menu and click **Refresh**. Refer Figure 12.13.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 12.1 for more details of each setting.
4. Click the **Retry** button.

Figure 12.13 ColdFire V1 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port

COM1 : Serial Port 1

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO USB

The P&E Cyclone PRO USB Connection setting permits a connection to Cyclone PRO USB devices. P&E Cyclone PRO USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone PRO USB** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next.

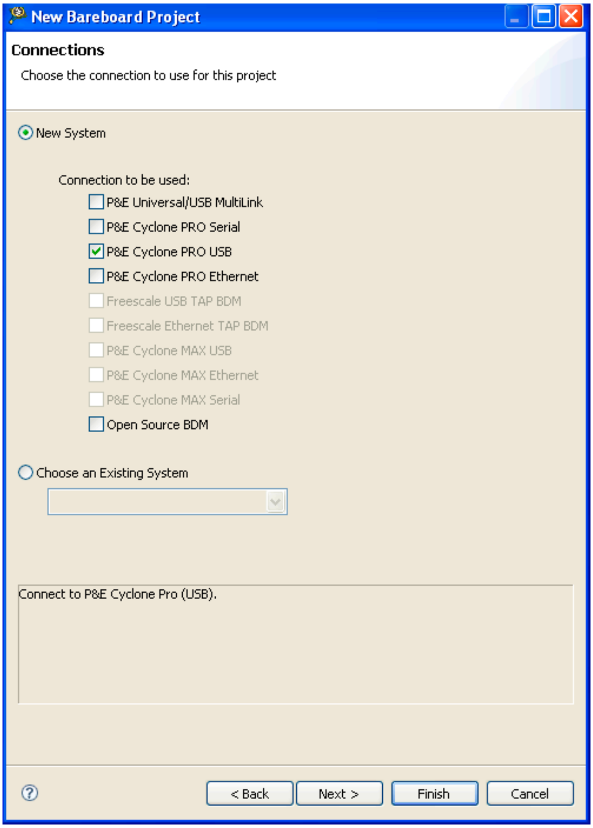
   The **Devices** page appears.

4. Expand the **ColdFire V1** tree control and select the derivative or board you would like to use. For example, select **ColdFire V1 > MCF51QE Family > MCF51QE128**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone PRO USB** checkbox. Refer Figure 12.14.

Figure 12.14 ColdFire V1 P&E Cyclone PRO USB Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☑ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (USB).

7. Click **Finish**.

The wizard creates a project for the ColdFire V1 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

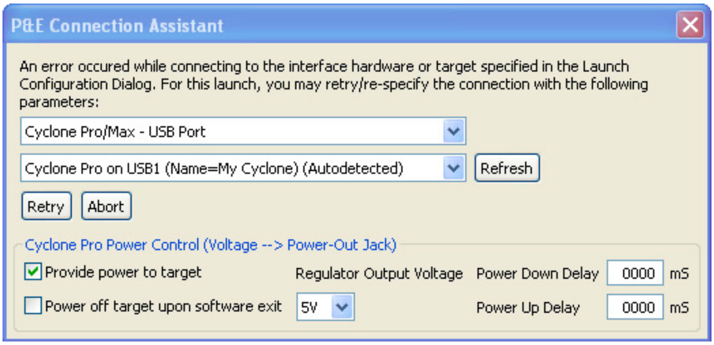
Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO USB** as your debugger connection:

1. Select **Cyclone Pro/Max – USB Port** from the first drop-down menu and click **Refresh**. Refer Figure 12.15.
2. Using the second drop-down menu, select the port on which the interface is connected.

3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 12.1 for more details of each setting.
4. Click the **Retry** button.

Figure 12.15 ColdFire V1 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Pro on USB1 (Name=My Cyclone) (Autodetected)

☑ Provide power to target

☐ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Cyclone PRO Ethernet

The P&E Cyclone PRO Ethernet Connection setting permits a connection to Cyclone PRO Ethernet devices. P&E Cyclone PRO Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone PRO Ethernet** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click Next.

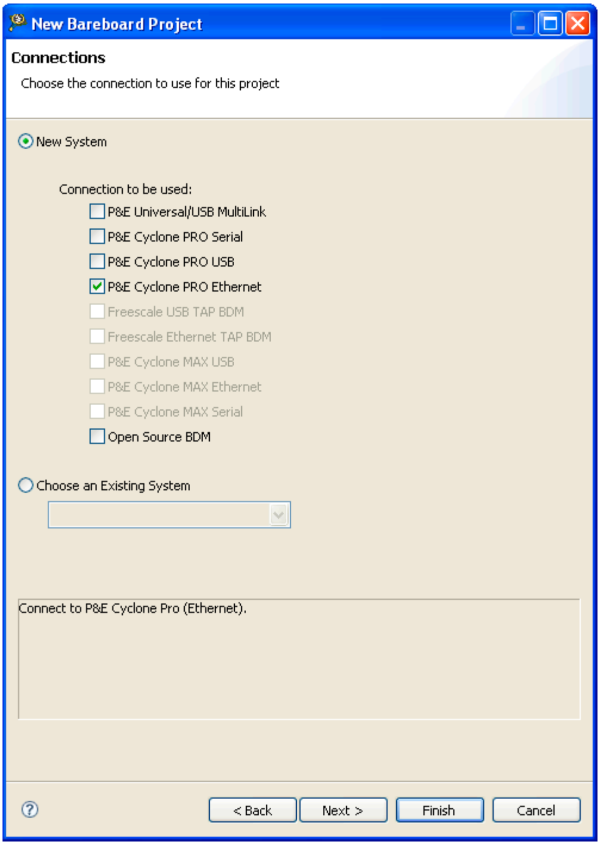
   The **Devices** page appears.

4. Expand the **ColdFire V1** tree control and select the derivative or board you would like to use. For example, select **ColdFire V1 > MCF51QE Family > MCF51QE128**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the P&E Cyclone PRO Ethernet checkbox. Refer Figure 12.16.

Figure 12.16 ColdFire V1 P&E Cyclone PRO Ethernet Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☑ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Pro (Ethernet).

7. Click **Finish**.

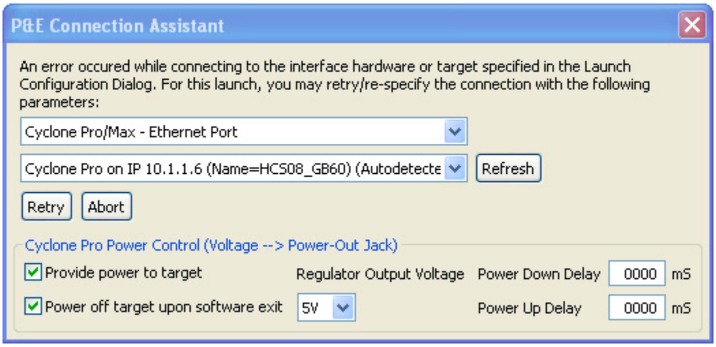
The wizard creates a project for the ColdFire V1 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone PRO Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down menu and click **Refresh**. Refer Figure 12.17.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **Cyclone Pro Power Control** panel to configure the power and delay settings. Refer to Table 12.1 for more details of each setting.
4. Click the **Retry** button.

Figure 12.17 ColdFire V1 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Pro on IP 10.1.1.6 (Name=HCS08_GB60) (Autodetecte

Cyclone Pro Power Control (Voltage --> Power-Out Jack)

☑ Provide power to target

☑ Power off target upon software exit

Regulator Output Voltage: 5V

Power Down Delay: 0000 mS

Power Up Delay: 0000 mS

P&E Open Source BDM

Freescale supplies certain development boards with an integrated debug circuit based on

Open Source BDM. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Pro or USB Multilink. The development board also derives its power from the USB Bus.

The Open Source BDM circuit design (OSBDM-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source BDM" connection. P&E has integrated the Open Source BDM support into the same connection that supports both the USB Multilink and the Cyclone Pro. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSBDM (albeit at a lower data rate).

The P&E Open Source BDM Connection setting permits a connection to Open Source BDM devices. Open Source BDM mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Open Source BDM** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

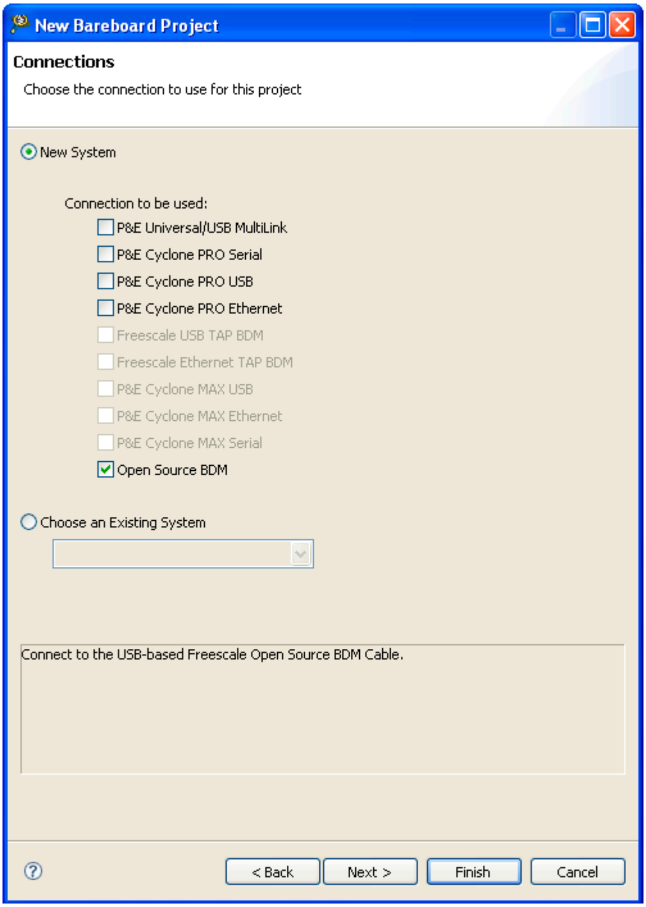
   The **Devices** page appears.

4. Expand the **ColdFire V1** tree control and select the derivative or board you would like to use. For example, select **ColdFire V1 > MCF51QE Family > MCF51QE128**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Open Source BDM** checkbox. Refer to Figure 12.18.

Figure 12.18 ColdFire V1 P&E Open Source BDM Selected



New Bareboard Project

Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☑ Open Source BDM

○ Choose an Existing System

Connect to the USB-based Freescale Open Source BDM Cable.

7. Click **Finish**.

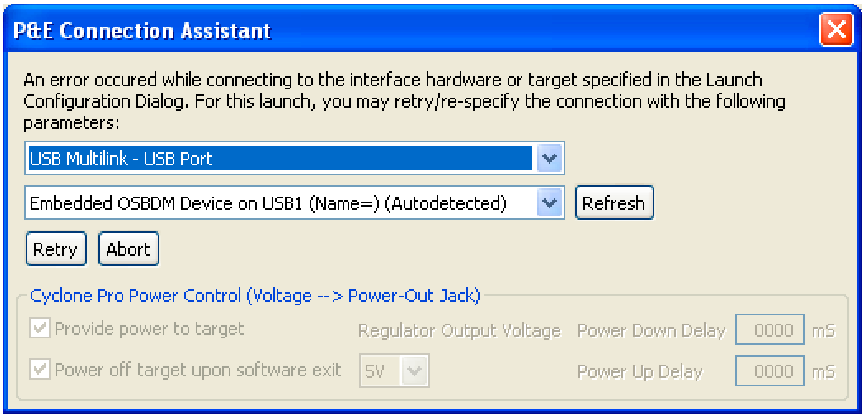
The wizard creates a project for the ColdFire V1 architecture according to your specifications. You can access the project from the **CodeWarrior Projects** view in the Workbench window.

Connection Assistant

The **P&E Connection Assistant** is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E Open Source BDM** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down menu and click **Refresh**. Refer to Figure 12.19.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Click the **Retry** button.

Figure 12.19 ColdFire V1 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

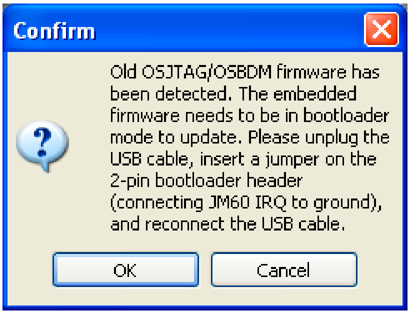
Embedded OSBDM Device on USB1 (Name=) (Autodetected)

Refresh
Retry
Abort
Cyclone Pro Power Control (Voltage --> Power-Out Jack)
☑ Provide power to target
☑ Power off target upon software exit
Regulator Output Voltage
5V
Power Down Delay 0000 mS
Power Up Delay 0000 mS

OSBDM Firmware Update

All Codewarrior IDE’s version 10.1 and higher have an automatic firmware update mechanism for built-in OSBDM hardware interfaces. Whenever an OSBDM-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSBDM firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

Figure 12.20 Old OSBDM Firmware Detected

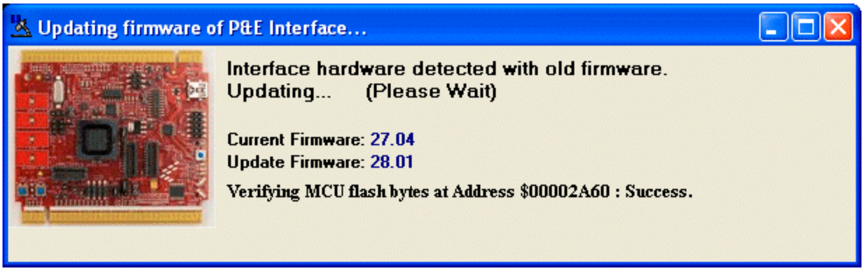


Confirm

Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSBDM device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSBDM-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSBDM device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSBDM device and click on the OK button. If done correctly, the automatic firmware update will occur.

Figure 12.21 OSBDM Firmware Updating

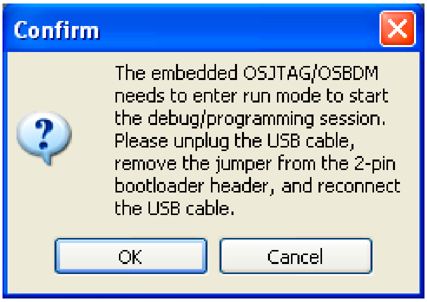


Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01
Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSBDM device must exit Bootloader mode and enter into Run mode.

Figure 12.22 Start OSBDM Run Mode



The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSBDM device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click on OK and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSBDM firmware. If for any reason you experience difficulty performing OSBDM firmware update, please visit www.pemicro.com/osbdm and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

CCS

This section discusses the option of different hardware probes supported by the CCS debugger protocol and their settings options. The supported hardware connections are:

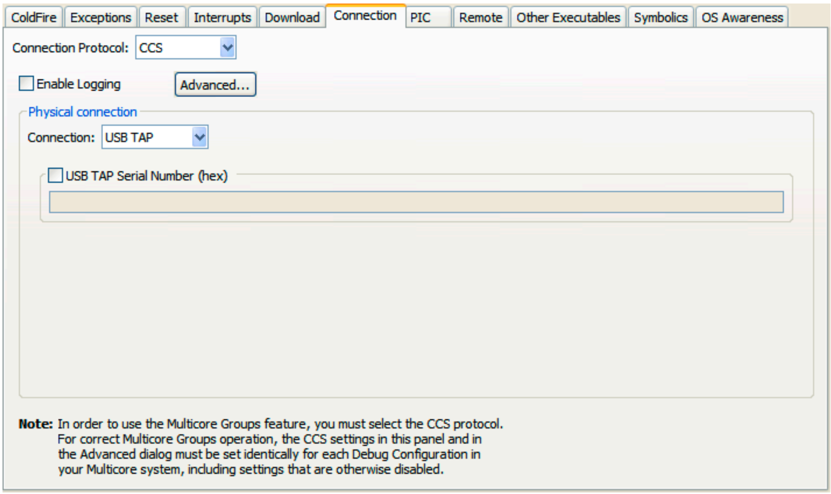
- Generic
- USB TAP
- Ethernet

NOTE The Generic connection is not supported for this release.

USB TAP

This option specifies that the physical connection uses USB TAP run control as the interface for debugging communications. The tab view displays the connection parameters this connection type uses (Figure 12.23).

Figure 12.23 CCS — USB Connection


ColdFire
Exceptions
Reset
Interrupts
Download
Connection
PIC
Remote
Other Executables
Symbolics
OS Awareness
Advanced...
Physical connection

Connection Protocol: CCS

☐ Enable Logging

Connection: USB TAP

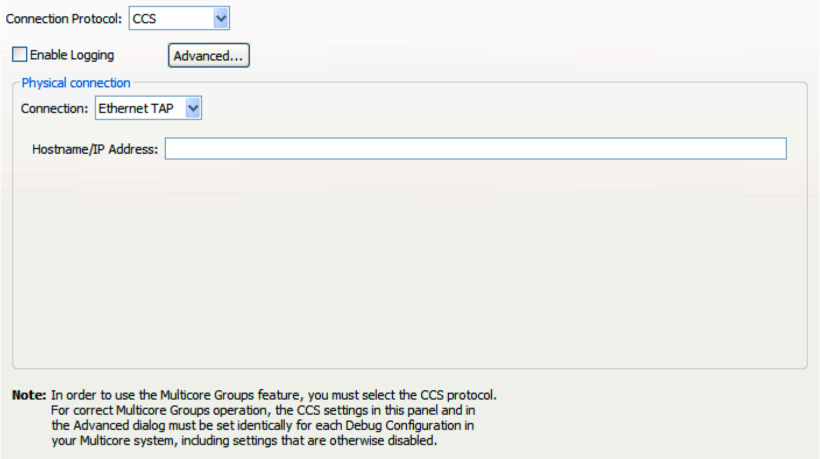
☐ USB TAP Serial Number (hex)

Note: In order to use the Multicore Groups feature, you must select the CCS protocol. For correct Multicore Groups operation, the CCS settings in this panel and in the Advanced dialog must be set identically for each Debug Configuration in your Multicore system, including settings that are otherwise disabled.

Ethernet

This option specifies that the physical connection uses Ethernet run control as the interface for debugging communications. The tab view displays the connection parameters this connection type uses ().

Figure 12.24 CCS — Ethernet TAP Connection



Connection Protocol: CCS

☐ Enable Logging

Connection: Ethernet TAP

Hostname/IP Address:

Note: In order to use the Multicore Groups feature, you must select the CCS protocol. For correct Multicore Groups operation, the CCS settings in this panel and in the Advanced dialog must be set identically for each Debug Configuration in your Multicore system, including settings that are otherwise disabled.

13

Connections — ColdFire V2/3/4

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the ColdFire V2/3/4 target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the **Launch Configuration Settings** panel. **Launch Configuration Settings** panel can be accessed by clicking on the **Edit** button located within the **Main** tab of the **Debug Configurations** dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with the ColdFire V2/3/4 device family.

The topics in this chapter are:

- Changing Connection in IDE
- P&E ColdFire Multilink/Cyclone MAX/OSBDM
- CCS
- CCS

Changing Connection in IDE

To change connection in the IDE, perform these steps.

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the tree control and select the derivative or board you would like to use.
5. Click **Next**.

   The **Connections** page appears.

6. Select the desired connection.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project according to your specifications. You can access the project from the **CodeWarrior Projects** view in the **Workbench** window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

P&E ColdFire Multilink/Cyclone MAX/OSBDM

This section describes the CFV2/3/4 P&E Connection options. The CFV2/3/4 P&E Connection setting permits a connection to CFV2/3/4 Freescale devices via P&E CFV2/3/4 Multilink/Cyclone Max/OSBDM hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

This topic describes all P&E CFV2/3/4 Multilink/Cyclone Max/OSBDM connection options, which are common to all P&E USB ColdFire Multilink or USB Multilink Universal, P&E Cyclone MAX Serial, P&E Cyclone MAX USB, P&E Cyclone MAX Ethernet, and P&E Open Source BDM connections.

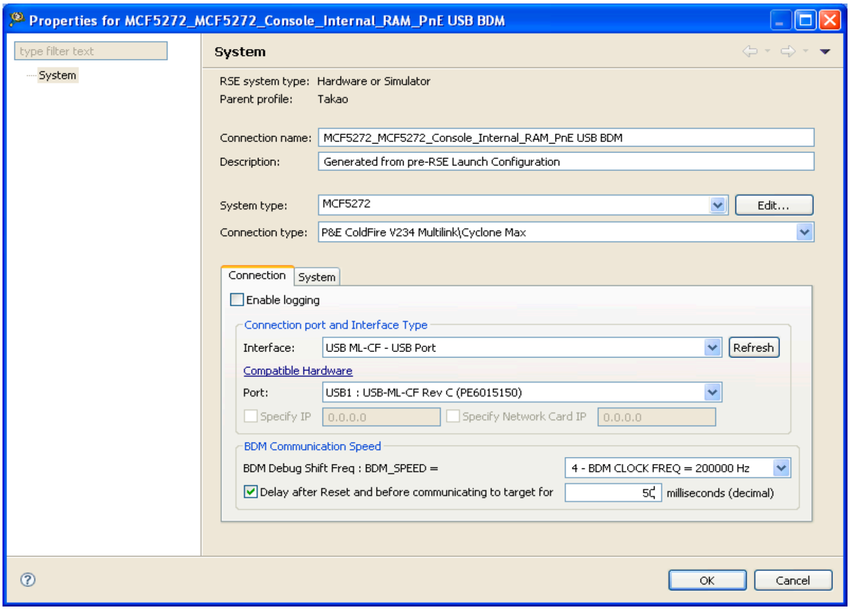
The options include:

- Changing P&E Connection Settings
- Connection Assistant
- View Register Files Options
- P&E ColdFire Multilink/Cyclone MAX/OSBDM Connection- Specific Options

Changing P&E Connection Settings

All connection settings for P&E hardware interfaces are configured in the **Launch Configurations** dialog box.

Figure 13.1 GDI — P&E CFV2/3/4 Multilink/Cyclone MAX/OSBDM Connection



Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

Table 13.1 describes the options for this view.

Table 13.1 Connection Parameter Options for P&E CFV2/3/4 Multilink/Cyclone MAX/ OSBDM

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB ColdFire Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone MAX - Serial Port • Cyclone MAX - USB Port • Cyclone MAX - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project. • OSBDM
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE**: If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help
(Cyclone Ethernet only) Specify IP	Use this option to specify the IP address of a Cyclone outside of the local network. Click on the checkbox to enable the textbox. This will also disable the port dropdown box. Currently supports IPv4 only.
(Cyclone Ethernet only) Specify Network Card IP	Use this option to specify the local network card IP address if there are multiple cards on your computer. Click on the checkbox to enable the textbox. Currently supports IPv4 only.

Table 13.1 Connection Parameter Options for P&E CFV2/3/4 Multilink/Cyclone MAX/ OSBDM (*continued*)

Option	Description
Refresh	.Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.
BDM Debug Shift Freq.	This option lets you to set the BDM shift clock speed of P&E's BDM interfaces. This integer value may be used to determine the speed of communications according to the following equations: Cyclone Max : (50000000/(2*N+5)) Hz USB-ML-CF : (1000000/(N+1)) Hz USB Multilink Universal: (1000000/(N+1)) Hz The value n should be between 0 and 31. This shift clock takes effect after the commands in the top of the programming algorithm are executed so that these commands can increase the target frequency and allow a faster shift clock. This clock can't generally exceed a div 4 of the processor bus frequency.
Delay After Reset	Specifies a delay after the programmer resets the target that we check to see if the part has properly gone into background debug mode. This is useful if the target has a reset driver which hold the MCU in reset after the programmer releases the reset line. The n value is a delay in milliseconds.

To change P&E Connections settings, perform these steps:

1. In the **CodeWarrior Projects** view, select the project for which you want to change the P&E Connections settings.

NOTE It is assumed that you have created a project and built it.

2. Select **Run > Debug Configurations** from the main menu bar of the IDE.

   The **Debug Configurations** dialog box appears.

3. Expand the **CodeWarrior Download** tree control in the left pane and select the launch configuration you want to debug.
4. Click the **Main** tab.

   The **Main** page appears in the area beneath the tabs.
5. Select a system within **Remote System** of which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the Freescale Eclipse Extensions Guide. Once a remote system is selected, click the **Edit** button. The **Launch Configuration Settings Panel** will appear.
6. Ensure that the **System Type** is the correct MCU you want to debug. Use the drop-down box or the **Edit** button to change this option.
7. In the **Connection Type** drop-down box, select P&E ColdFire V234 Multilink\Cyclone MAX. The P&E connections settings will appear below.
8. Click **Refresh** to scan valid interface and port.

   Valid interfaces and ports appear in the **Interface** and **Port** drop-down lists in the **Connection** port and **Interface Type** group.
9. Select a supported interface from the **Interface** drop-down list.
10. Select a supported port from the **Port** drop-down list.

NOTE The port displayed may vary depending on the interface. For example, if you select interface as Cyclone MAX - Serial Port, the available port option is COM1 : Serial Port 1.

11. Select a BDM speed using the drop-down box in the **BDM Debug Shift Frequency**.

NOTE If you select a fast BDM speed, there may be scenarios where you may have difficulty communicating with your target device. You may want to experimentally select a BDM speed that is fast and yet have a good communication with your target.

12. Click on the **Delay after Reset** checkbox and enter the desired delay (in milliseconds) in the text box.

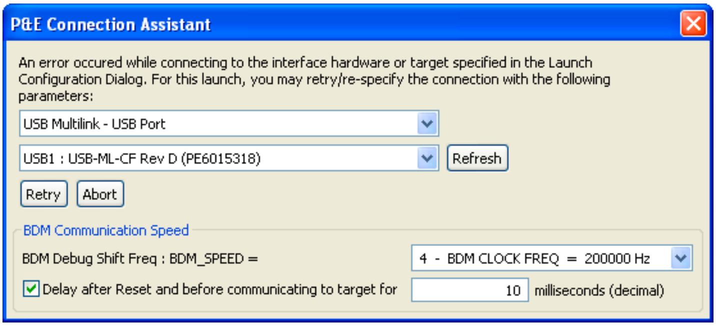
    This option specifies the time interval to wait between resetting and communicating the target device.
13. Click the **OK** to save changes to the P&E Connections settings. The **Launch Configuration Settings Panel** dialog box will close.
14. Click on the **Close** button to close the **Debug Configuration** dialog box.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug but the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E CFV2/3/4 Multilink/Cyclone MAX/OSBDM** as your debugger connection:

1. Choose the **P&E** device that you are using from the first drop-down menu and click **Refresh**. Refer Figure 13.2.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 13.2 CFV2/3/4 Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB1 : USB-ML-CF Rev D (PE6015318)

BDM Communication Speed

BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz

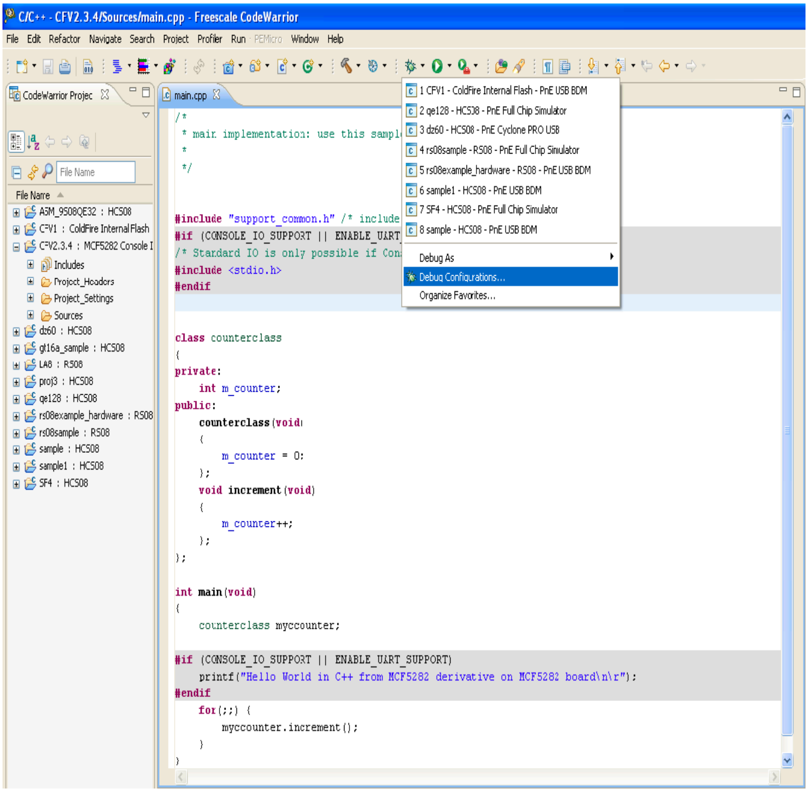
☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

Launch Configuration Settings

To set the launch configurations for the debugger:

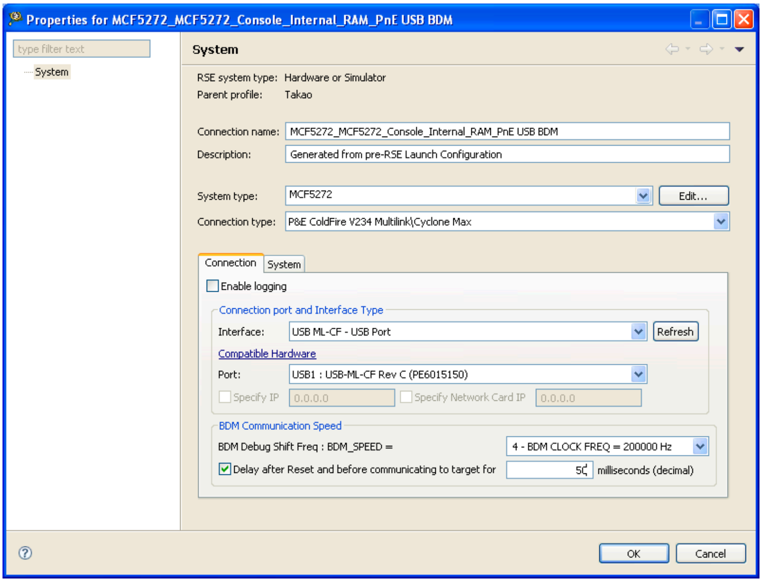
1. Find the debugger icon and click on the drop-down arrow to bring up the debugger menu. Refer Figure 13.2.
2. Select **Debug Configurations**.
3. In the left column, select the project download type for which you would like to set the launch configurations. Refer Figure 13.3
4. In the right column, click on the **Debugger** tab.
5. Set your configurations and click **Debug** to start the debugger.

Figure 13.3 Debugger Drop-down Menu


C/C++ - CFV2.3.4/Sources/main.cpp - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run PEMicro Window Help
CodeWarrior Projec
File Name
File Name
ASM_9S08QE32 : HCS08
CFV1 : ColdFire Internal Flash
CFV2.3.4 : MCF5282 Console I
Includes
Project_Headers
Project_Settings
Sources
dz60 : HCS08
gt16a_sample : HCS08
LAB : RS08
proj3 : HCS08
qe128 : HCS08
rs08example_hardware : RS08
rs08sample : RS08
sample : HCS08
sample1 : HCS08
SF4 : HCS08
main.cpp
1 CFV1 - ColdFire Internal Flash - PnE USB BDM
2 qe128 - HCS08 - PnE Full Chip Simulator
3 dz60 - HCS08 - PnE Cyclone PRO USB
4 rs08sample - RS08 - PnE Full Chip Simulator
5 rs08example_hardware - RS08 - PnE USB BDM
6 sample1 - HCS08 - PnE USB BDM
7 SF4 - HCS08 - PnE Full Chip Simulator
8 sample - HCS08 - PnE USB BDM
Debug As
Debug Configurations...
Organize Favorites...

```
/*
 * main implementation: use this sampl
 *
 */


#include "support_common.h" /* include
#if (CONSOLE_IO_SUPPORT || ENABLE_UART
/* Standard IO is only possible if Con
#include <stdio.h>
#endif


class counterclass
{
private:
    int m_counter;
public:
    counterclass(void)
    {
        m_counter = 0;
    };
    void increment(void)
    {
        m_counter++;
    };
};

int main(void)
{
    counterclass myccounter;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++ from MCF5282 derivative on MCF5282 board\n\r");
#endif
    for(;;) {
        myccounter.increment();
    }
}
```

Figure 13.4 Debugger Configuration Settings Dialog Box


Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
type filter text
System
System
Edit...
Connection
System
Refresh
Compatible Hardware
OK
Cancel

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

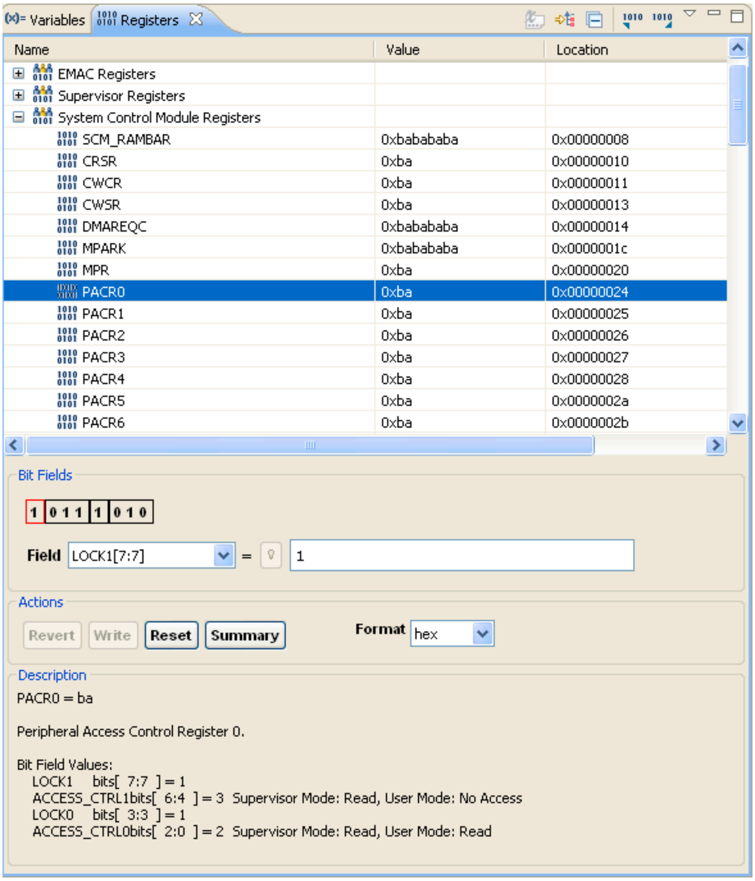
Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0 ☐ Specify Network Card IP 0.0.0.0

BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

View Register Files Options

The Register Files tab in the debugger gives you option of viewing and editing the register files. If register files are available for the device that you have chosen, the **Registers** tab in the debugger (see Figure 13.5) is populated.

Figure 13.5 Debug Register File Tab



Name	Value	Location
⊞ EMAC Registers		
⊞ Supervisor Registers		
⊟ System Control Module Registers		
SCM_RAMBAR	0xbababab a	0x00000008
CRSR	0xba	0x00000010
CWCR	0xba	0x00000011
CWSR	0xba	0x00000013
DMAREQC	0xbababab a	0x00000014
MPARK	0xbababab a	0x0000001c
MPR	0xba	0x00000020
PACR0	0xba	0x00000024
PACR1	0xba	0x00000025
PACR2	0xba	0x00000026
PACR3	0xba	0x00000027
PACR4	0xba	0x00000028
PACR5	0xba	0x0000002a
PACR6	0xba	0x0000002b

Field LOCK1[7:7] = 1

Format hex

```
PACR0 = ba

Peripheral Access Control Register 0.

Bit Field Values:
   LOCK1     bits[ 7:7 ] = 1
   ACCESS_CTRL1bits[ 6:4 ] = 3  Supervisor Mode: Read, User Mode: No Access
   LOCK0     bits[ 3:3 ] = 1
   ACCESS_CTRL0bits[ 2:0 ] = 2  Supervisor Mode: Read, User Mode: Read
```

To view the **Register Files** of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window
2. Select the **Registers** tab on the right side of the debugging window, or select the **Window menu > Show View > Registers** to open the **Register** window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the **Registers** tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit

field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The **Actions** box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The **Description** box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

You can modify each bit by selecting it in the drop-down box under the **Bit** field, or by clicking the bit on the **Bit** field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

P&E ColdFire Multilink/Cyclone MAX/ OSBDM Connection- Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB ColdFire Multilink or USB Multilink Universal
- P&E Cyclone MAX Serial
- P&E Cyclone MAX USB
- P&E Cyclone MAX Ethernet
- P&E Open Source BDM

P&E USB ColdFire Multilink or USB Multilink Universal

The P&E USB ColdFire Multilink or USB Multilink Universal Connection setting permits a connection to USB ColdFire Multilink devices. P&E USB ColdFire Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources. The Multilink connection option can also be used to connect a USB Multilink Universal to your interface.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E USB ColdFire Multilink** or **USB Multilink Universal** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

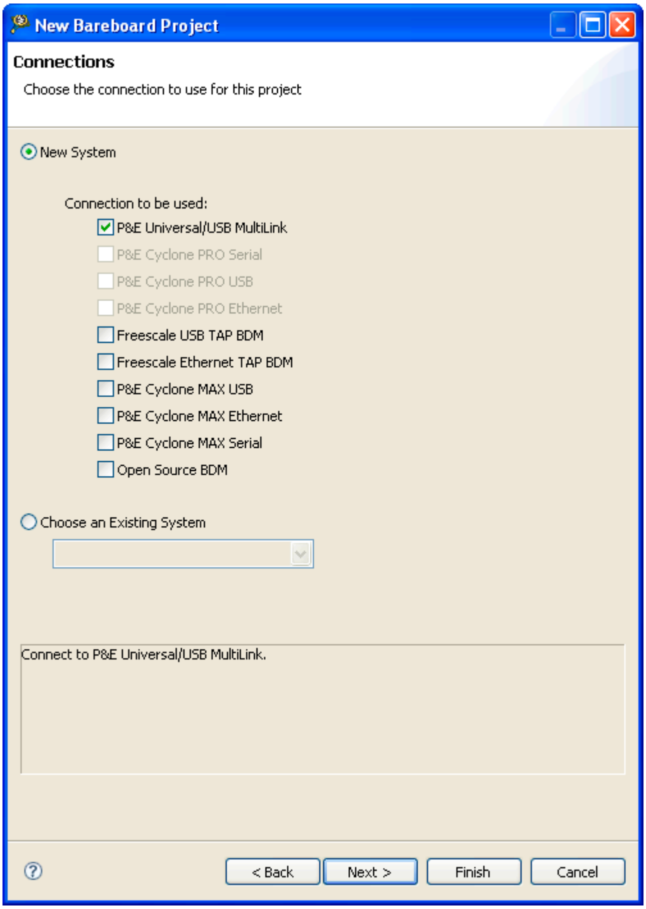
   The **Devices** page appears.

4. Expand the **ColdFire V2/3/4** tree control and select the derivative or board you would like to use. For example, select **ColdFire V2 > MCF5225X Family > MCF52259**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Universal/USB Multilink** checkbox. Refer to Figure 13.6.

Figure 13.6 P&E USB ColdFire Multilink Selected



● New System

Connection to be used:

☑ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Universal/USB MultiLink.

7. Click **Finish**.

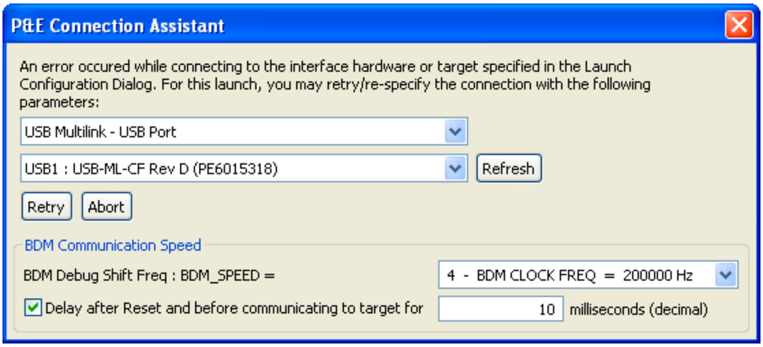
The wizard creates a project for the ColdFire architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E USB ColdFire Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 13.7.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 13.7 ColdFire Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB1 : USB-ML-CF Rev D (PE6015318)

BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX Serial

The P&E Cyclone MAX Serial Connection setting permits a connection to Cyclone MAX Serial devices. P&E Cyclone MAX Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Serial** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

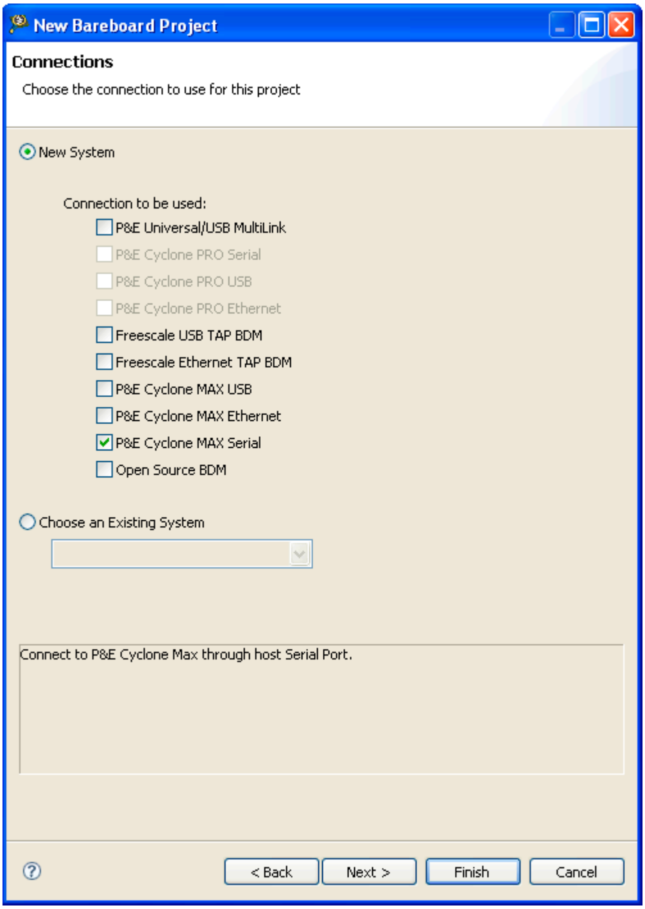
3. Click **Next**.

   The **Devices** page appears.

4. Expand the **ColdFire** tree control and select the derivative or board you would like to use. For example, select **ColdFire V2 > MCF5225X Family > MCF52259**.
5. Click **Next**.

   The **Connections** page appears.
6. Check the **P&E Cyclone MAX Serial** checkbox. Refer to Figure 13.8.

Figure 13.8 ColdFire P&E Cyclone MAX Serial Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☑ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Max through host Serial Port.

7. Click **Finish**.

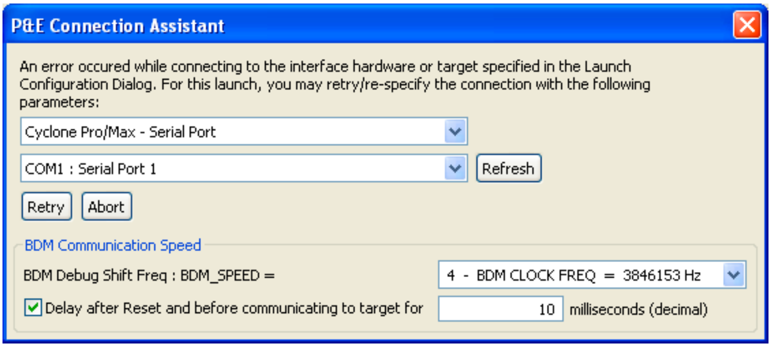
The wizard creates a project for the ColdFire architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone MAX Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down list and click **Refresh**. Refer to Figure 13.9.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 13.9 ColdFire Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port

COM1 : Serial Port 1

BDM Communication Speed

BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 3846153 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX USB

The P&E Cyclone MAX USB Connection setting permits a connection to Cyclone MAX USB devices. P&E Cyclone MAX USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone MAX USB** as the debugger connection:

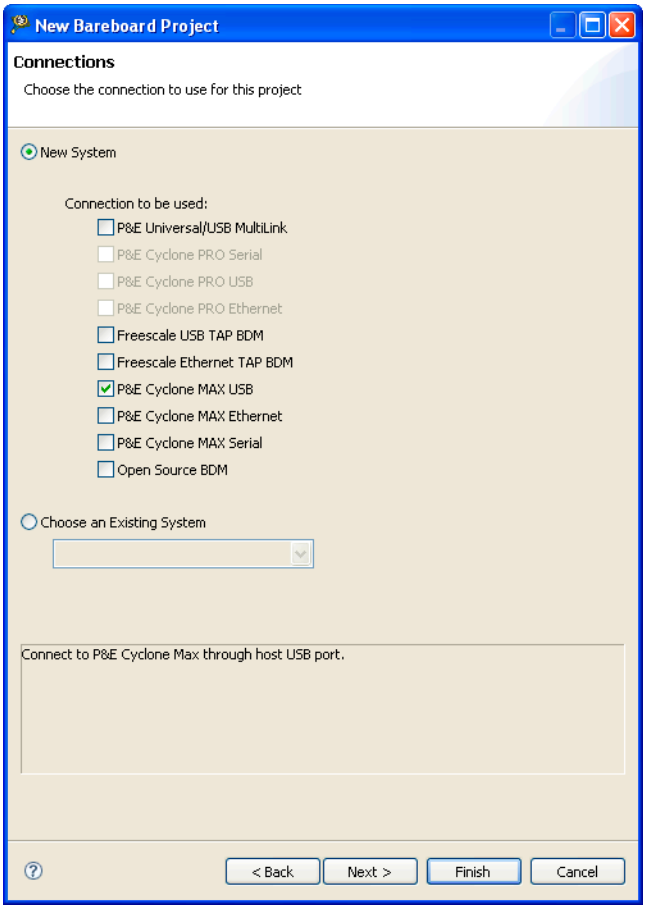
1. Select **Project > Change Device/Connection** from the IDE menu bar. The **Device/Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.
4. Expand the **ColdFire** tree control and select the derivative or board you would like to use. For example, select **ColdFire V2 > MCF5225X Family > MCF52259**.
5. Click Next. The **Connections** page appears.
6. Check the **P&E Cyclone MAX USB** checkbox. Refer to Figure 13.10.

Figure 13.10 ColdFire P&E Cyclone MAX USB Selected


New Bareboard Project
Connections
Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☑ P&E Cyclone MAX USB
☐ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Max through host USB port.

< Back
Next >
Finish
Cancel

7. Click **Finish**.

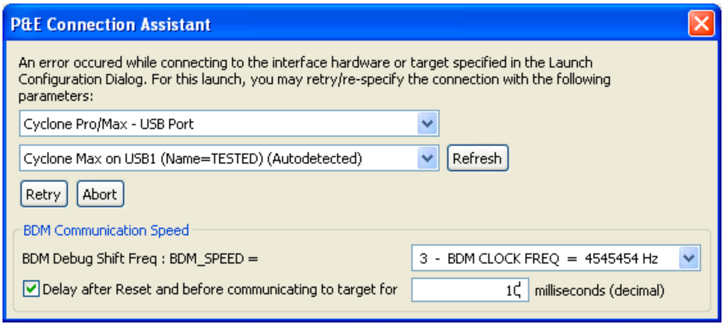
The wizard creates a project for the **ColdFire** architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the P&E Cyclone MAX USB as your debugger connection:

1. Select **Cyclone Pro/Max – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 13.11.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details of each setting.
4. Click the **Retry** button.

Figure 13.11 ColdFire Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Max on USB1 (Name=TESTED) (Autodetected)

BDM Debug Shift Freq : BDM_SPEED = 3 - BDM CLOCK FREQ = 4545454 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX Ethernet

The P&E Cyclone MAX Ethernet Connection setting permits a connection to Cyclone MAX Ethernet devices. P&E Cyclone MAX Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Ethernet** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

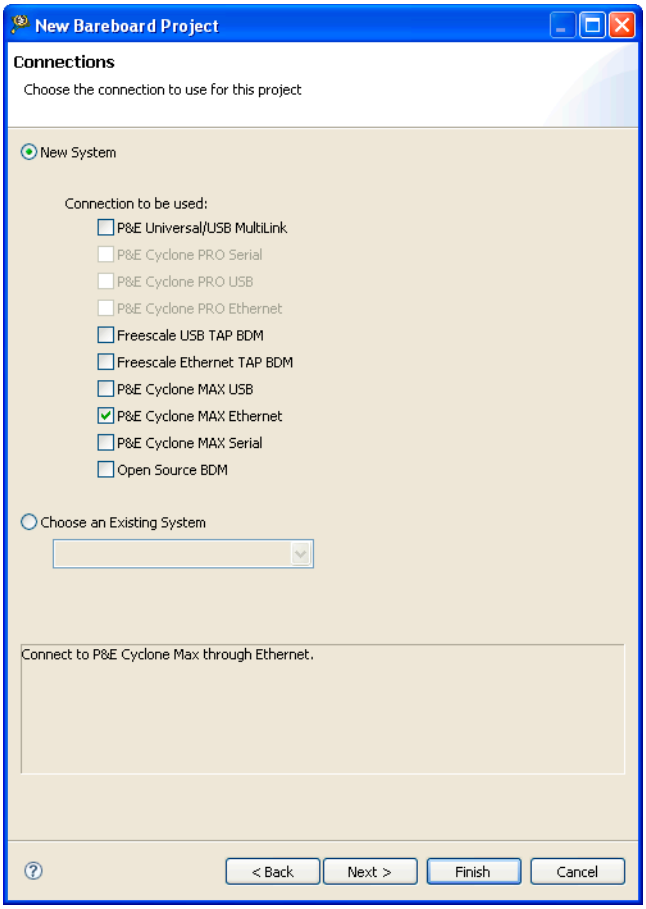
   The **Devices** page appears.
4. Expand the **ColdFire** tree control and select the derivative or board you would like to use. For example, select **ColdFire V2 > MCF5225X Family > MCF52259**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone MAX Ethernet** checkbox. Refer to Figure 13.12.

Figure 13.12 ColdFire P&E Cyclone MAX Ethernet Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

☐ P&E Universal/USB MultiLink
☐ P&E Cyclone PRO Serial
☐ P&E Cyclone PRO USB
☐ P&E Cyclone PRO Ethernet
☐ Freescale USB TAP BDM
☐ Freescale Ethernet TAP BDM
☐ P&E Cyclone MAX USB
☑ P&E Cyclone MAX Ethernet
☐ P&E Cyclone MAX Serial
☐ Open Source BDM

○ Choose an Existing System

Connect to P&E Cyclone Max through Ethernet.

7. Click **Finish**.

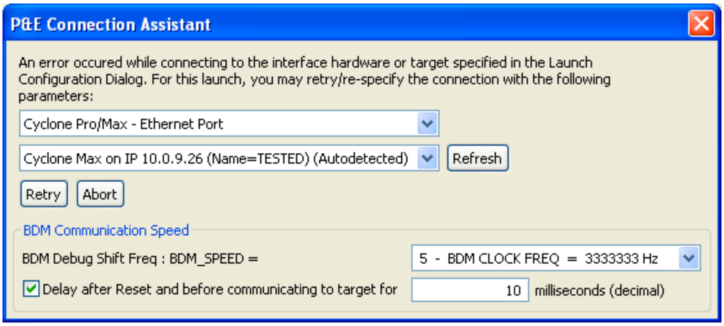
The wizard creates a project for the **ColdFire** architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The **P&E Connection Assistant** is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E Cyclone MAX Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down list and click **Refresh**. Refer to Figure 13.13.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details of each setting.
4. Click the **Retry** button.

Figure 13.13 ColdFire Connection Assistant Interface Selected


P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Max on IP 10.0.9.26 (Name=TESTED) (Autodetected)

BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 3333333 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Open Source BDM

Freescale supplies certain development boards with an integrated debug circuit based on Open Source BDM. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Max or USB ColdFire Multilink. The development board also derives its power from the USB Bus.

The Open Source BDM circuit design (OSBDM-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source BDM" connection. P&E has integrated the Open Source BDM support into the same connection that supports both the USB ColdFire Multilink and the Cyclone Max. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSBDM (albeit at a lower data rate).

The P&E Open Source BDM Connection setting permits a connection to Open Source BDM devices. Open Source BDM mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Open Source BDM** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar. The **Device/ Connection Change** wizard appears.
2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

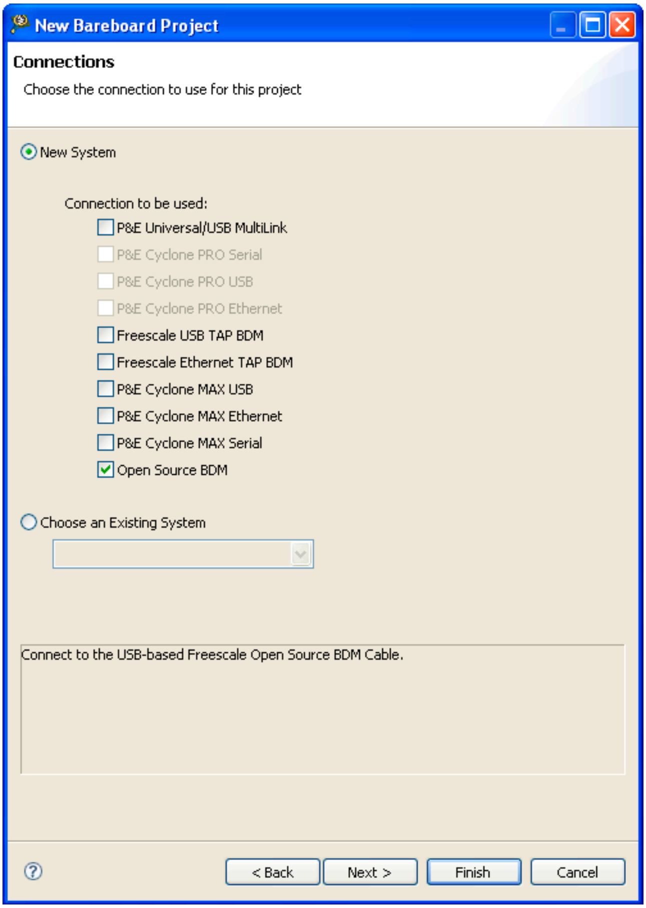
NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.
4. Expand the **ColdFire** tree control and select the derivative or board you would like to use. For example, select **ColdFire V2 > MCF5225X Family > MCF52259**.
5. Click **Next**.

   The **Connections** page appears.
6. Check the **P&E Open Source BDM** checkbox. Refer to Figure 13.14.

Figure 13.14 ColdFire P&E Open Source BDM Selected



Connections

Choose the connection to use for this project

● New System

Connection to be used:

- ☐ P&E Universal/USB MultiLink
- ☐ P&E Cyclone PRO Serial
- ☐ P&E Cyclone PRO USB
- ☐ P&E Cyclone PRO Ethernet
- ☐ Freescale USB TAP BDM
- ☐ Freescale Ethernet TAP BDM
- ☐ P&E Cyclone MAX USB
- ☐ P&E Cyclone MAX Ethernet
- ☐ P&E Cyclone MAX Serial
- ☑ Open Source BDM

○ Choose an Existing System

Connect to the USB-based Freescale Open Source BDM Cable.

7. Click **Finish**.

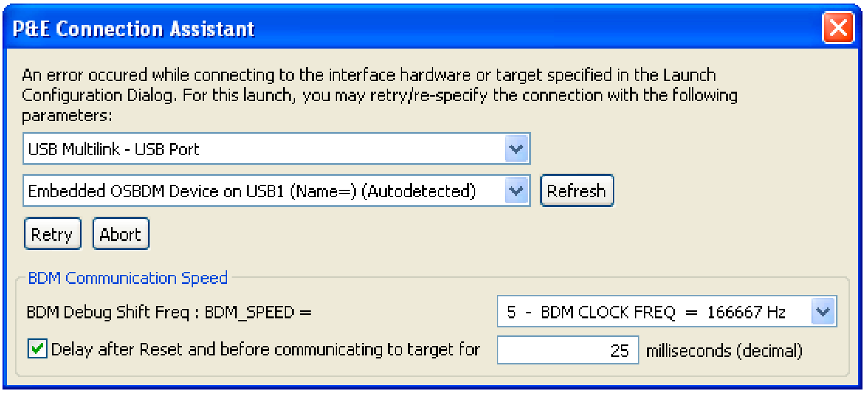
The wizard creates a project for the **ColdFire** architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select **P&E Open Source BDM** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click Refresh. Refer to Figure 13.15.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 13.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 13.15 ColdFire Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

Embedded OSBDM Device on USB1 (Name=) (Autodetected)

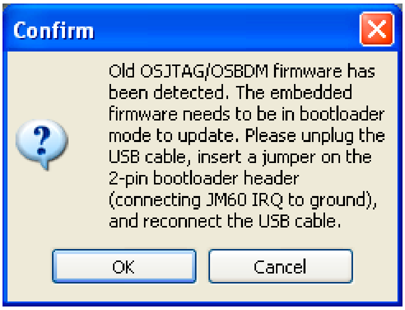
BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 166667 Hz

☑ Delay after Reset and before communicating to target for 25 milliseconds (decimal)

OSBDM Firmware Update

All Codewarrior IDE’s version 10.1 and higher have an automatic firmware update mechanism for built-in OSBDM hardware interfaces. Whenever an OSBDM-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSBDM firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

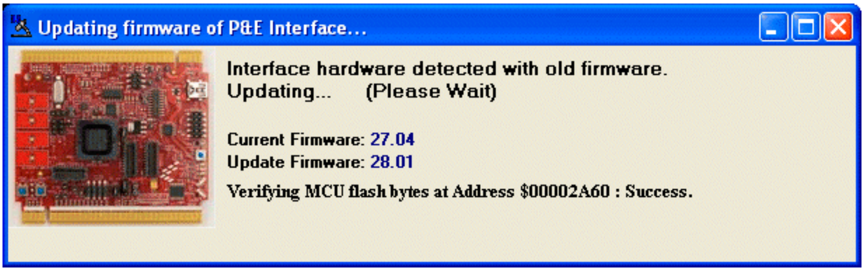
Figure 13.16 Old OSBDM Firmware Detected



Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSBDM device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSBDM-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSBDM device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSBDM device and click on the OK button. If done correctly, the automatic firmware update will occur.

Figure 13.17 OSBDM Firmware Updating



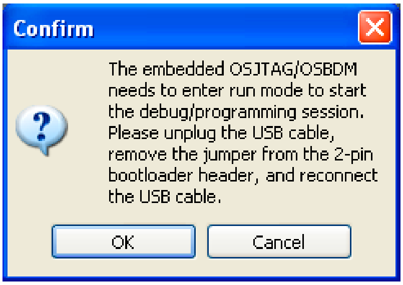
Updating firmware of P&E Interface...

Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01
Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSBDM device must exit Bootloader mode and enter into Run mode.

Figure 13.18 Start OSBDM Run Mode



Confirm

The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSBDM device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click on OK and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSBDM firmware. If for any reason you experience difficulty performing OSBDM firmware update, please visit www.pemicro.com/osbdm and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

CCS

This section discusses the option of different hardware probes supported by the CCS debugger protocol and their settings options. The supported hardware connections are:

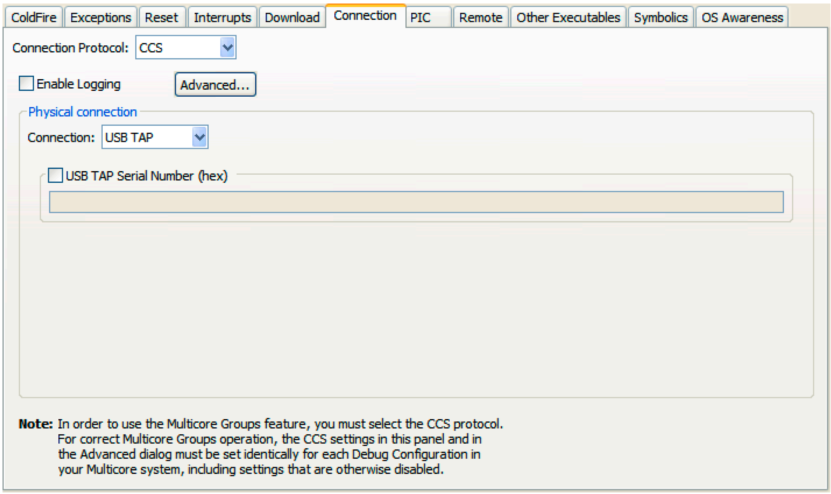
- Generic
- USB TAP
- Ethernet

NOTE The Generic connection is not supported for this release.

USB TAP

This option specifies that the physical connection uses USB TAP run control as the interface for debugging communications. The tab view displays the connection parameters this connection type uses (Figure 13.19).

Figure 13.19 CCS — USB Connection



Connection Protocol: CCS

☐ Enable Logging

Physical connection

Connection: USB TAP

☐ USB TAP Serial Number (hex)

Note: In order to use the Multicore Groups feature, you must select the CCS protocol. For correct Multicore Groups operation, the CCS settings in this panel and in the Advanced dialog must be set identically for each Debug Configuration in your Multicore system, including settings that are otherwise disabled.

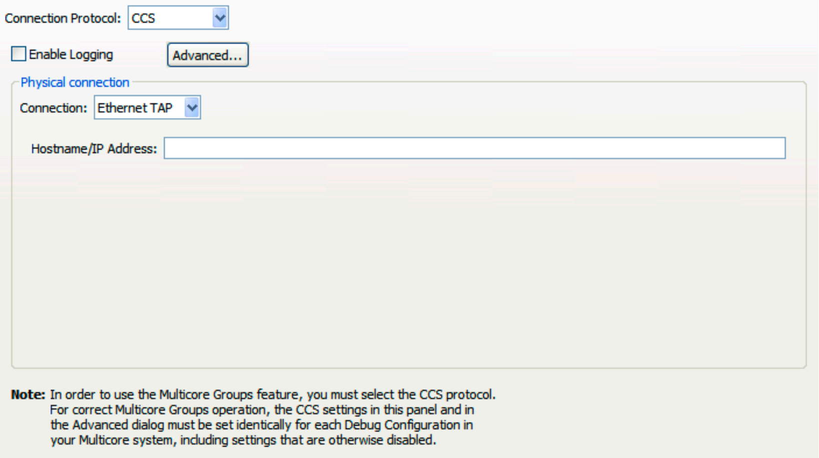
You use the option **USB TAP Serial Number (hex)** to enter an optional serial number for the USB hardware probe. Enter the serial number into the text box as a hexadecimal value.

NOTE For more information on the **Advanced** option presented on these views, refer to the topic *Connection Tab — ColdFire* in the *Working with the Debugger* chapter,

Ethernet

This option specifies that the physical connection uses Ethernet run control as the interface for debugging communications. The tab view displays the connection parameters this connection type uses ().

Figure 13.20 CCS — Ethernet TAP Connection



Connection Protocol: CCS

☐ Enable Logging

Connection: Ethernet TAP

Hostname/IP Address:

Note: In order to use the Multicore Groups feature, you must select the CCS protocol. For correct Multicore Groups operation, the CCS settings in this panel and in the Advanced dialog must be set identically for each Debug Configuration in your Multicore system, including settings that are otherwise disabled.

Enter the IP address of the Ethernet TAP into the **Hostname/IP Address** text box. Enter this value as a dotted decimal number, such as 127.0.0.1.

NOTE For more information on the **Advanced** option presented on these views, refer to the topic *Connection Tab — ColdFire* in the *Working with the Debugger* chapter.

14

Connections — Power Architecture

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the Power Architecture target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the Launch Configuration Settings panel. **Launch Configuration Settings** panel can be accessed by clicking on the **Edit** button located within the Main tab of the **Debug Configurations** dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with the Power Architecture device family.

The topics in this chapter are:

- Changing Connection in IDE
- P&E PPCNEXUS Multilink/Cyclone MAX/OSJTAG

Changing Connection in IDE

To change connection in the IDE, perform these steps.

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the tree control and select the derivative or board you would like to use.
5. Click **Next**.

   The **Connections** page appears.

6. Select the desired connection.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project according to your specifications. You can access the project from the CodeWarrior Projects view in the Workbench window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

P&E PPCNEXUS Multilink/Cyclone MAX/OSJTAG

This section describes PPCNEXUS P&E Multilink/Cyclone Max Connection options. The PPCNEXUS P&E Multilink/Cyclone Max Connection setting permits a connection to PPCNEXUS Freescale devices via P&E Multilink/Cyclone Max hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

Connection Options

This topic describes all P&E PPCNEXUS Multilink/Cyclone Max/OSJTAG connection options, which are common to all P&E USB PPCNexus Multilink or USB Multilink Universal, P&E Cyclone MAX Serial, P&E Cyclone MAX USB, P&E Cyclone MAX Ethernet, and P&E Open Source JTAG connections.

The options include:

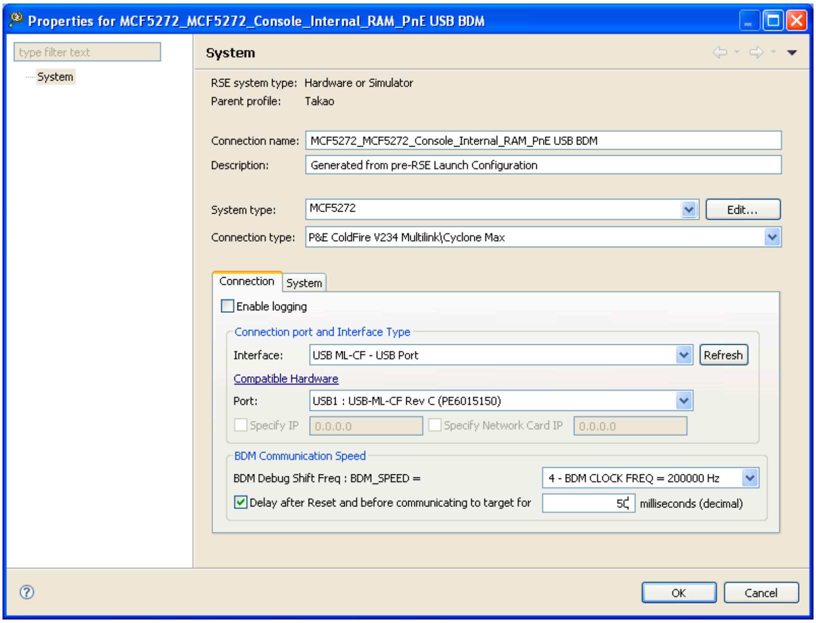
- Changing P&E Connections Settings
- Connection Assistant
- Launch Configuration Settings
- Active Mode Menu Options
- Advanced Programming/Debug Options

- [View Register Files Options]
- [P&E PPCNexus Multilink/Cyclone MAX/OSJTAG Connection- Specific Options]

Changing P&E Connections Settings

All connection settings for P&E hardware interfaces are configured in the **Launch Configurations** dialog box.

Figure 14.1 GDI — P&E PPCNEXUS Multilink/Cyclone MAX/OSJTAG Connection



Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0 ☐ Specify Network Card IP 0.0.0.0

BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

Table Table 14.1 describes the options for this view.

Table 14.1 Connection Parameter Options for P&E PPCNEXUS Multilink/Cyclone MAX/ OSJTAG

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB PPCNexus Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone MAX - Serial Port • Cyclone MAX - USB Port • Cyclone MAX - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project. • OSJTAG
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE**: If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help.
(Cyclone Ethernet only) Specify IP	Use this option to specify the IP address of a Cyclone outside of the local network. Click on the checkbox to enable the textbox. This will also disable the port dropdown box. Currently supports IPv4 only.
(Cyclone Ethernet only) Specify Network Card IP	Use this option to specify the local network card IP address if there are multiple cards on your computer. Click on the checkbox to enable the textbox. Currently supports IPv4 only.

Table 14.1 Connection Parameter Options for P&E PPCNEXUS Multilink/Cyclone MAX/ OSJTAG

Option	Description
Refresh	Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.
BDM Debug Shift Freq.	This option lets you to set the BDM shift clock speed of P&E's BDM interfaces. This integer value may be used to determine the speed of communications according to the following equations: Cyclone Max: (50000000/(2*N+5)) Hz USB-ML-PPCNEXUS: (1000000/(N+1)) Hz USB Multilink Universal: (1000000/(N+1)) Hz The value n should be between 0 and 31. This shift clock takes effect after the commands in the top of the programming algorithm are executed so that these commands can increase the target frequency and allow a faster shift clock. This clock can't generally exceed a div 4 of the processor bus frequency.
Delay After Reset	Specifies a delay after the programmer resets the target that we check to see if the part has properly gone into background debug mode. This is useful if the target has a reset driver which hold the MCU in reset after the programmer releases the reset line. The n value is a delay in milliseconds.

To change P&E Connections settings, perform these steps:

1. In the **CodeWarrior Projects** view, select the project for which you want to change the P&E Connections settings.

NOTE It is assumed that you have created a project and built it.

2. Select **Run > Debug Configurations** from the main menu bar of the IDE.

   The **Debug Configurations** dialog box appears.

3. Expand the **CodeWarrior Download** tree control in the left pane and select the launch configuration you want to debug.
4. Click the **Main** tab.

   The **Main** page appears in the area beneath the tabs.
5. Select a system within **Remote System** of which you would like to use to debug. You could create a new system by clicking the **New** button. For more details about creating a new remote system, refer to the topic Remote System Explorer in the Freescale Eclipse Extensions Guide. Once a remote system is selected, click the **Edit** button. The **Launch Configuration Settings Panel** will appear.
6. Ensure that the **System Type** is the correct MCU you want to debug. Use the drop-down box or the **Edit** button to change this option.
7. In the **Connection Type** drop-down box, select P&E PPCNexus Multilink\Cyclone MAX. The P&E connections settings will appear below.
8. Click **Refresh** to scan valid interface and port.

   Valid interfaces and ports appear in the **Interface** and **Port** drop-down lists in the **Connection** port and **Interface Type** group.
9. Select a supported interface from the **Interface** drop-down list.
10. Select a supported port from the **Port** drop-down list.

NOTE The port displayed may vary depending on the interface. For example, if you select interface as Cyclone MAX - Serial Port, the available port option is COM1 : Serial Port 1.

11. Select a BDM speed using the drop-down box in the **BDM Debug Shift Frequency**.

NOTE If you select a fast BDM speed, there may be scenarios where you may have difficulty communicating with your target device. You may want to experimentally select a BDM speed that is fast and yet have a good communication with your target.

12. Click on the **Delay after Reset** checkbox and enter the desired delay (in milliseconds) in the text box.

    This option specifies the time interval to wait between resetting and communicating the target device.
13. Click the **OK** to save changes to the P&E Connections settings. The **Launch Configuration Settings Panel** dialog box will close.

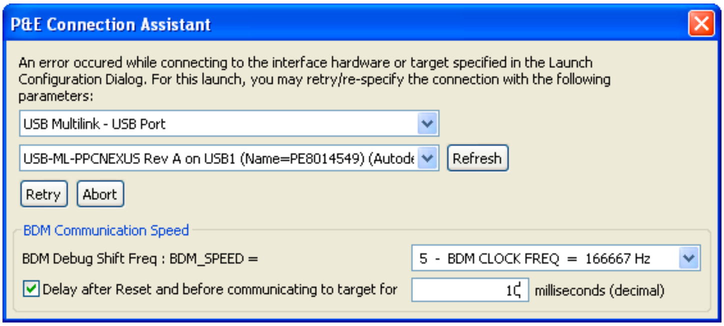
Click on the **Close** button to close the **Debug Configuration** dialog box.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug but the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E PPCNEXUS Multilink/Cyclone Max/OSJTAG** as your debugger connection:

1. Choose the **P&E** device that you are using from the first drop-down menu and click **Refresh**. Refer to Figure 14.2.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 14.2 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-PPCNEXUS Rev A on USB1 (Name=PE8014549) (Autode

BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 166667 Hz

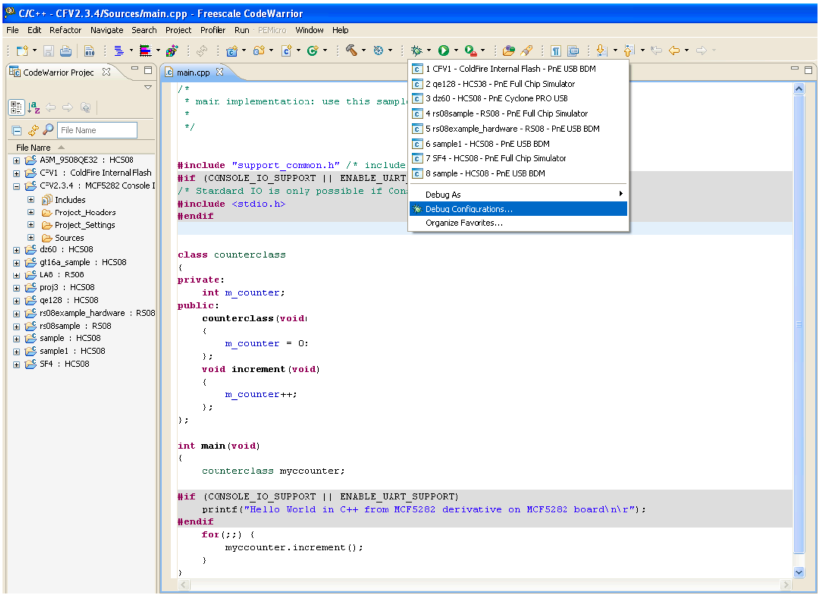
☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

Launch Configuration Settings

To set the launch configurations for the debugger:

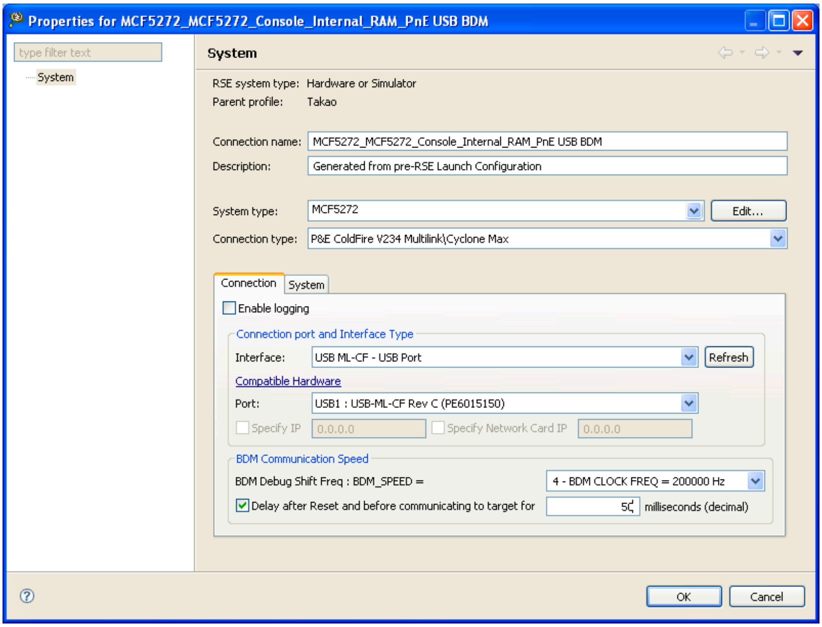
1. Find the debugger icon and click on the drop-down arrow to bring up the debugger menu. Refer to Figure 14.2.
2. Select **Debug Configurations**.
3. In the left column, select the project download type for which you would like to set the launch configurations. Refer to Figure 14.3
4. In the right column, set your configurations and click **Debug** to start the debugger.

Figure 14.3 Debugger Drop-down Menu


C/C++ - CFV2.3.4/Sources/main.cpp - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run PEMicro Window Help
CodeWarrior Projec
File Name
File Name
ASM_9S08QE32 : HCS08
CFV1 : ColdFire Internal Flash
CFV2.3.4 : MCF5282 Console I
Includes
Project_Headers
Project_Settings
Sources
dz60 : HCS08
gt16a_sample : HCS08
LA8 : RS08
proj3 : HCS08
qe128 : HCS08
rs08example_hardware : RS08
rs08sample : RS08
sample : HCS08
sample1 : HCS08
SF4 : HCS08
main.cpp
1 CFV1 - ColdFire Internal Flash - PnE USB BDM
2 qe128 - HCS08 - PnE Full Chip Simulator
3 dz60 - HCS08 - PnE Cyclone PRO USB
4 rs08sample - RS08 - PnE Full Chip Simulator
5 rs08example_hardware - RS08 - PnE USB BDM
6 sample1 - HCS08 - PnE USB BDM
7 SF4 - HCS08 - PnE Full Chip Simulator
8 sample - HCS08 - PnE USB BDM
Debug As
Debug Configurations...
Organize Favorites...

```
/*
 * main implementation: use this sampl
 *
 */


#include "support_common.h" /* include
#if (CONSOLE_IO_SUPPORT || ENABLE_UART
/* Standard IO is only possible if Con
#include <stdio.h>
#endif


class counterclass
{
private:
    int m_counter;
public:
    counterclass(void)
    {
        m_counter = 0;
    };
    void increment(void)
    {
        m_counter++;
    };
};

int main(void)
{
    counterclass myccounter;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++ from MCF5282 derivative on MCF5282 board\n\r");
#endif
    for(;;) {
        myccounter.increment();
    }
}
```

Figure 14.4 Debugger Configuration Settings Dialog Box



Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

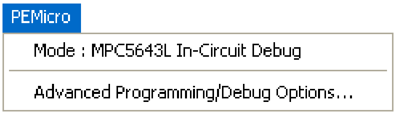
BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

type filter text
System
Edit...
Connection
System
Refresh
Compatible Hardware
OK
Cancel

Active Mode Menu Options

When the microprocessor is connected, the active mode menu shows the name of the microprocessor and gives you the access to the Advanced Programming/Debug Options. When the microprocessor is not connected, the menu is not available.

Figure 14.5 Additional Connection Menu Options

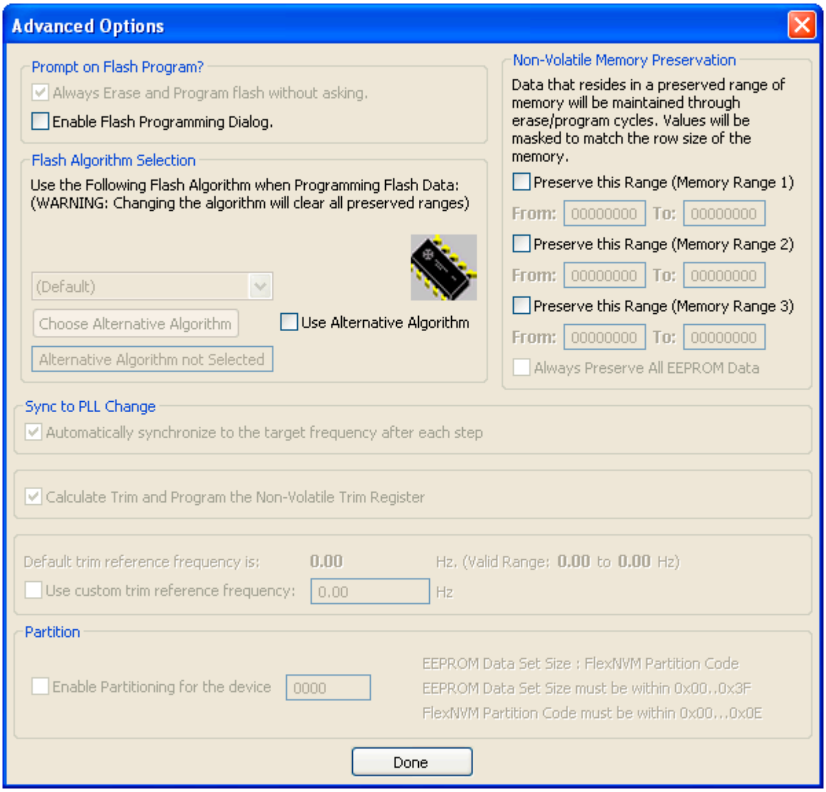


PEMicro
Mode : MPC5643L In-Circuit Debug
Advanced Programming/Debug Options...

Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the **Advanced Options** dialog box, where you can configure the software settings for the flash programming procedure.

Figure 14.6 Advanced Options Dialog Box



Prompt on Flash Program?
☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)
(Default)
Choose Alternative Algorithm
☐ Use Alternative Algorithm
Alternative Algorithm not Selected

Non-Volatile Memory Preservation
Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.
☐ Preserve this Range (Memory Range 1)
From: 00000000 To: 00000000
☐ Preserve this Range (Memory Range 2)
From: 00000000 To: 00000000
☐ Preserve this Range (Memory Range 3)
From: 00000000 To: 00000000
☐ Always Preserve All EEPROM Data

Sync to PLL Change
☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 0.00 Hz. (Valid Range: 0.00 to 0.00 Hz)
☐ Use custom trim reference frequency: 0.00 Hz

Partition
☐ Enable Partitioning for the device 0000
EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Enable Flash Programming Dialog

Setting the Enable Flash Programming dialog box lets you view the steps taken by the Flash Programmer.

Non-Volatile Memory Preservation

You have the option of preserving up to three independent ranges of non-volatile memory (on devices with EEPROM, the EEPROM array may optionally be preserved as well). Ranges that are designated as “preserved” are read before an erase, and reprogrammed immediately afterwards, thereby preserving the data in these ranges. Any attempt to program data into a preserved range is ignored. When entering an address into the preserved range field (hexadecimal input is expected), the values are masked according to the row size of the device. This ensures that the reprogramming of preserved data does not cause any conditions that disturb programming.

Alternative Algorithm Functionality

Once you create a project for a specific PPCNexus microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in the <CW_Install>/MCU/bin/plugins/support/EPPC/gdi/P&E directory

However, you can override the default algorithm via the Alternative Algorithm function, located in the Advanced Programming/Debug Options menu. This feature can be used to select a custom programming algorithm, or select another one of P&E's many programming algorithms for use with a specific project.

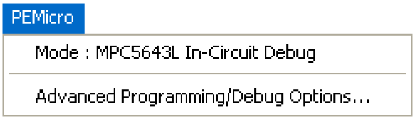
TIP Selecting a wrong programming algorithm may damage you device, lead to under/over programming situations, or simply not program portions of the project file.

Therefore it is recommended to use the default algorithm unless there is a compelling reason to do otherwise.

Use these steps to override the default algorithm:

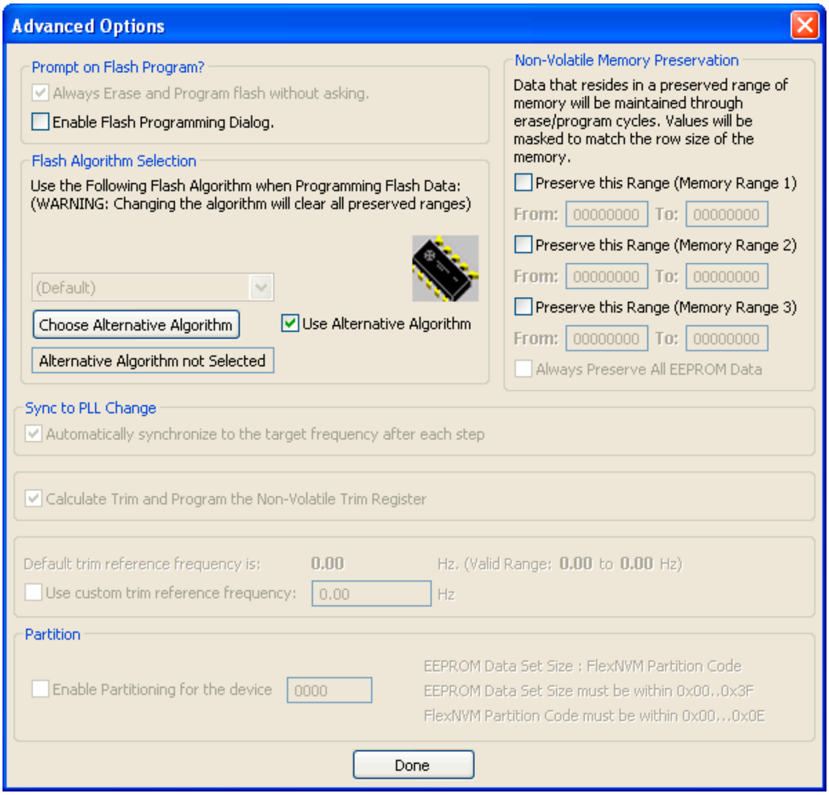
1. Select the **Advanced Programming/Debug Options** selection from the **PEMicro** menu.

Figure 14.7 Advanced Programming/Debug Options Menu Selection


PEMicro
Mode : MPC5643L In-Circuit Debug
Advanced Programming/Debug Options...

2. Check the Use Alternative Algorithm checkbox.

Figure 14.8 Advanced Options - Alternative Algorithm Checkbox



Advanced Options

Prompt on Flash Program?
☑ Always Erase and Program flash without asking.
☐ Enable Flash Programming Dialog.

Flash Algorithm Selection
Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)
(Default)
Choose Alternative Algorithm
☑ Use Alternative Algorithm
Alternative Algorithm not Selected

Non-Volatile Memory Preservation
Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.
☐ Preserve this Range (Memory Range 1)
From: 00000000 To: 00000000
☐ Preserve this Range (Memory Range 2)
From: 00000000 To: 00000000
☐ Preserve this Range (Memory Range 3)
From: 00000000 To: 00000000
☐ Always Preserve All EEPROM Data

Sync to PLL Change
☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 0.00 Hz. (Valid Range: 0.00 to 0.00 Hz)
☐ Use custom trim reference frequency: 0.00 Hz

Partition
☐ Enable Partitioning for the device 0000
EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Done

3. Click the Choose Alternative Algorithm button, which lets you browse for an alternative algorithm.

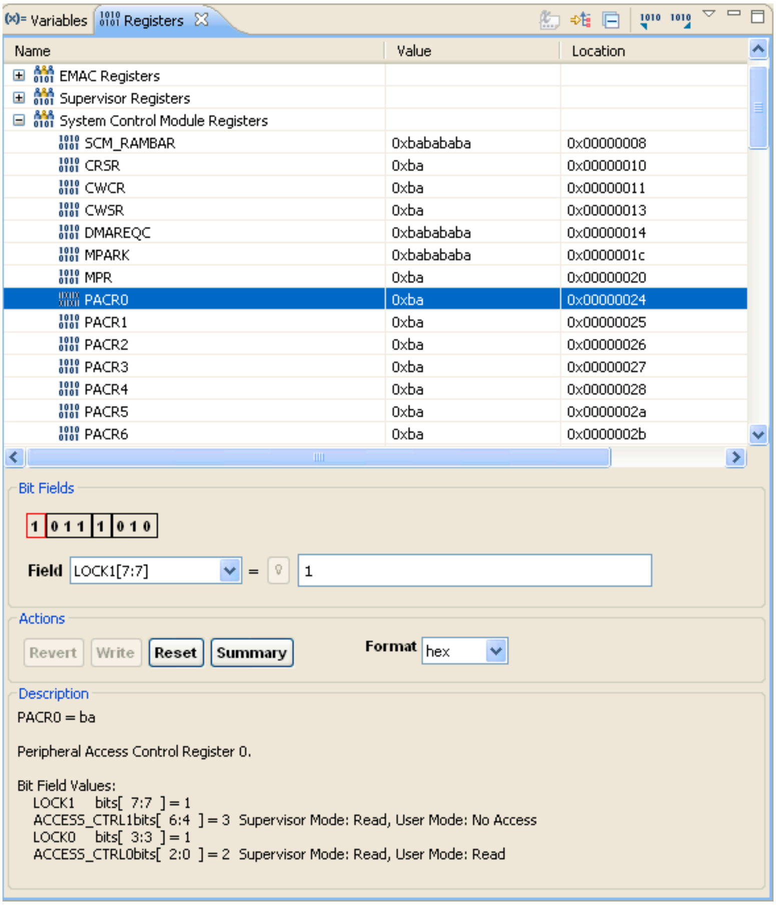
4. Once you select the alternative algorithm, the name of the algorithm along with its full path appears in the text field below the Choose Alternative Algorithm button.

At this point, the current project performs all future Flash programming operations using the alternative algorithm. You may revert to the default algorithm at any time by clearing the Use Alternative Algorithm checkbox.

View Register Files Options

The Register Files tab in the debugger gives you option of viewing and editing the register files. If register files are available for the device that you have chosen, the **Registers** tab in the debugger (see Figure 14.9) is populated.

Figure 14.9 Debug Register File Tab



Name	Value	Location
⊞ EMAC Registers		
⊞ Supervisor Registers		
⊟ System Control Module Registers		
SCM_RAMBAR	0xbabababa	0x00000008
CRSR	0xba	0x00000010
CWCR	0xba	0x00000011
CWSR	0xba	0x00000013
DMAREQC	0xbabababa	0x00000014
MPARK	0xbabababa	0x0000001c
MPR	0xba	0x00000020
PACR0	0xba	0x00000024
PACR1	0xba	0x00000025
PACR2	0xba	0x00000026
PACR3	0xba	0x00000027
PACR4	0xba	0x00000028
PACR5	0xba	0x0000002a
PACR6	0xba	0x0000002b

1 | 0 1 1 | 1 | 0 1 0

Field LOCK1[7:7] = 1

Format hex

```
PACR0 = ba

Peripheral Access Control Register 0.

Bit Field Values:
   LOCK1    bits[  7:7  ] = 1
   ACCESS_CTRL1bits[  6:4  ] = 3  Supervisor Mode: Read, User Mode: No Access
   LOCK0    bits[  3:3  ] = 1
   ACCESS_CTRL0bits[  2:0  ] = 2  Supervisor Mode: Read, User Mode: Read
```

To view the **Register Files** of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window
2. Select the **Registers** tab on the right side of the debugging window, or select the **Window > Show View > Registers** to open the Register window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the **Registers** tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit

field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The Actions box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The Description box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

You can modify each bit by selecting it in the drop-down box under the Bit field, or by clicking the bit on the Bit field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

P&E PPCNexus Multilink/Cyclone MAX/ OSJTAG Connection- Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB PPCNexus Multilink or USB Multilink Universal
- P&E Cyclone MAX Serial
- P&E Cyclone MAX USB
- P&E Cyclone MAX Ethernet
- P&E Open Source JTAG

P&E USB PPCNexus Multilink or USB Multilink Universal

The P&E USB PPCNexus Multilink or USB Multilink Universal Connection setting permits a connection to USB PPCNexus Multilink devices. P&E USB PPCNexus Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources. The MultiLink connection option can also be used to connect a USB Multilink Universal to your interface.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E USB PPCNexus Multilink** or **USB Multilink Universal** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Power Architecture** tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E USB PPCNexus Multilink** checkbox.
7. Click **Finish**.

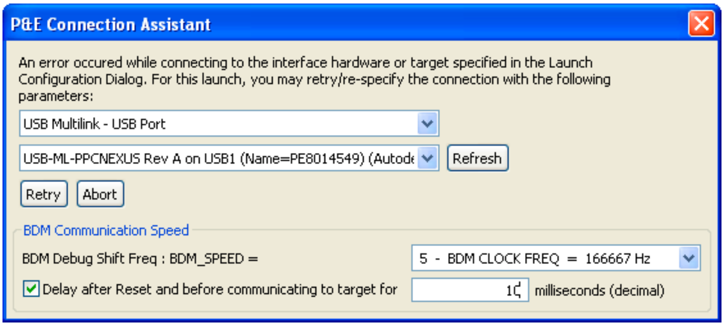
The wizard creates a project for Power Architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E USB PPCNexus Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 14.10.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details of each setting.
4. Click the **Retry** button.

Figure 14.10 Power Architecture Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-PPCNEXUS Rev A on USB1 (Name=PE8014549) (Autod

BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 166667 Hz

☑ Delay after Reset and before communicating to target for 1 milliseconds (decimal)

P&E Cyclone MAX Serial

The P&E Cyclone MAX Serial Connection setting permits a connection to Cyclone MAX Serial devices. P&E Cyclone MAX Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Serial** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Power Architecture** tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone MAX Serial** checkbox. If P&E Cyclone MAX Serial is not available, then select **P&E USB PPCNexus Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**. Refer to Changing P&E Connections Settings for information on how to change the settings.

7. Click **Finish**.

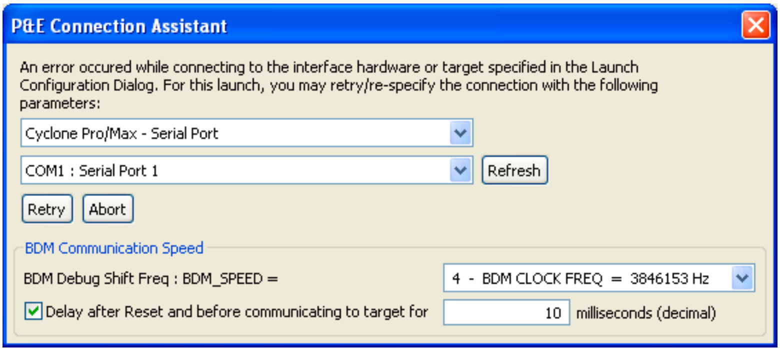
The wizard creates a project for Power Architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone MAX Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down list and click **Refresh**. Refer to Figure 14.11.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 14.11 Power Architecture Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port

COM1 : Serial Port 1

BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 3846153 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX USB

The P&E Cyclone MAX USB Connection setting permits a connection to Cyclone MAX USB devices. P&E Cyclone MAX USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone MAX USB** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Power Architecture** tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone MAX USB** checkbox. If P&E Cyclone MAX USB is not available, then select **P&E USB PPCNexus Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**. Refer to Changing P&E Connections Settings for information on how to change the settings.

7. Click **Finish**.

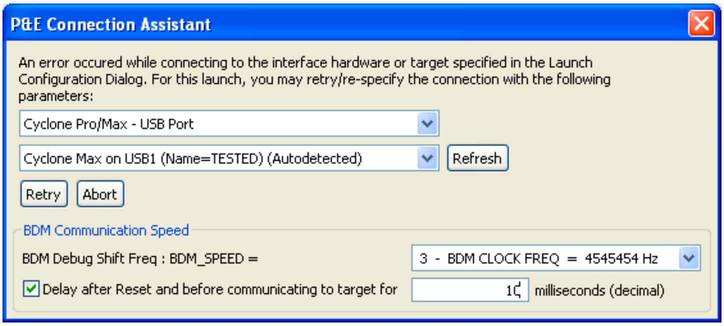
The wizard creates a project for **Power Architecture** according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone MAX USB** as your debugger connection:

1. Select **Cyclone Pro/MAX – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 14.12.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details of each setting.
4. Click the **Retry** button.

Figure 14.12 Power Architecture Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Max on USB1 (Name=TESTED) (Autodetected)

BDM Communication Speed

BDM Debug Shift Freq : BDM_SPEED = 3 - BDM CLOCK FREQ = 4545454 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX Ethernet

The P&E Cyclone MAX Ethernet Connection setting permits a connection to Cyclone MAX TCP-IP devices. P&E Cyclone MAX Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Ethernet** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Power Architecture** tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone MAX Ethernet** checkbox. If P&E Cyclone MAX Ethernet is not available, then select **P&E USB PPCNexus Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**.

Refer to Changing P&E Connections Settings for information on how to change the settings.

7. Click **Finish**.

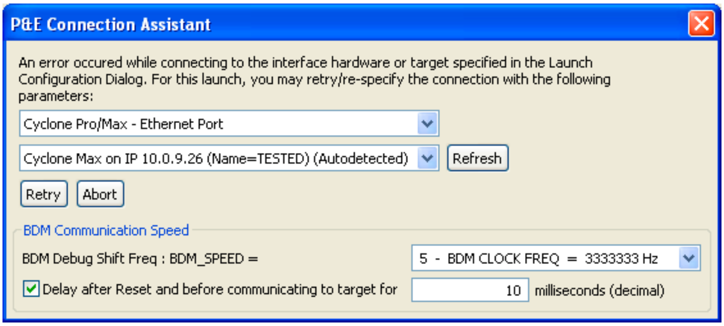
The wizard creates a project for Power Architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the **P&E Cyclone MAX Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down list and click **Refresh**. Refer to Figure 14.13.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details of each setting.
4. Click the **Retry** button.

Figure 14.13 Power Architecture Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Max on IP 10.0.9.26 (Name=TESTED) (Autodetected)

BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 3333333 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Open Source JTAG

Freescale supplies certain development boards with an integrated debug circuit based on Open Source JTAG. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Max or USB PPCNexus Multilink. The development board also derives its power from the USB Bus.

The Open Source JTAG circuit design (OSJTAG-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source TAG" connection. P&E has integrated the Open Source BDM support into the same connection that supports both the USB ColdFire Multilink and the Cyclone Max. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSJTAG (albeit at a lower data rate).

The Open Source JTAG Connection setting permits a connection to Open Source JTAG devices. Open Source JTAG mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Open Source JTAG** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Power Architecture** tree control and select the derivative or board you would like to use. For example, select **MPC56xx > MPC56xxL Family > MPC5643L**.

5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Open Source JTAG** checkbox.

7. Click **Finish**.

The wizard creates a project for Power Architecture according to your specifications. You can access the project from the CodeWarrior projects view in the Workbench window.

Connection Assistant

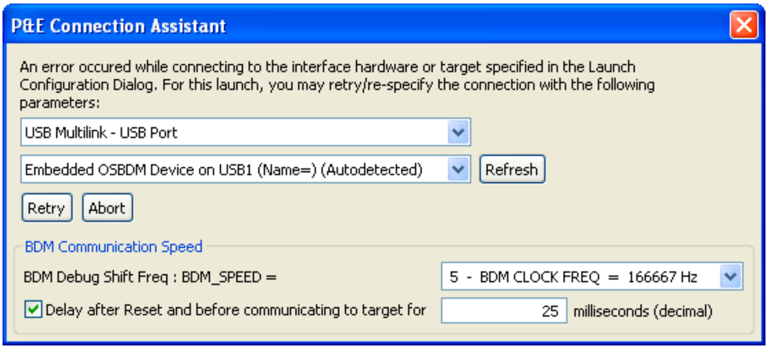
The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select **P&E Open Source JTAG** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 14.14.

2. Using the second drop-down list, select the port on which the interface is connected.

3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 14.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 14.14 Power Architecture Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

Embedded OSBDM Device on USB1 (Name=) (Autodetected)

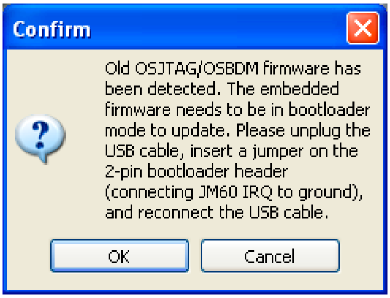
BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 166667 Hz

☑ Delay after Reset and before communicating to target for 25 milliseconds (decimal)

OSJTAG Firmware Update

All Codewarrior IDE’s version 10.1 and higher have an automatic firmware update mechanism for built-in OSJTAG hardware interfaces. Whenever an OSJTAG-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSJTAG firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

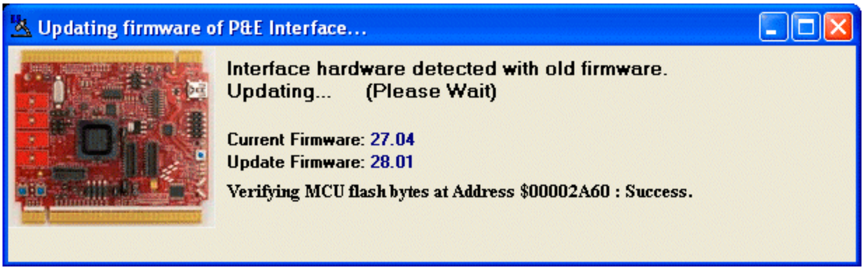
Figure 14.15 Old OSJTAG Firmware Detected



Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSJTAG device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSJTAG-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSJTAG device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSJTAG device and click on the OK button. If done correctly, the automatic firmware update will occur.

Figure 14.16 OSJTAG Firmware Updating

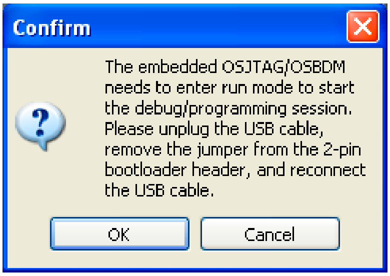


Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01
Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSJTAG device must exit Bootloader mode and enter into Run mode.

Figure 14.17 Start OSJTAG Run Mode



The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSJTAG device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click on OK and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSJTAG firmware. If for any reason you experience difficulty performing OSJTAG firmware update, please visit www.pemicro.com/osbdm and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

15

Connections — Kinetis

This chapter describes the features and settings of the connections that interface the CodeWarrior debugger with the Kinetis target board.

For the IDE to communicate with the target hardware, you must specify several key items: the debugger protocol, a connection type, and any connection parameters. You can enter these items using options in the **Launch Configuration Settings** panel. Launch Configuration Settings panel can be accessed by clicking on the **Edit** button located within the **Main** tab of the **Debug Configurations** dialog box. These options are:

- The **Connection Type** option determines what debugger protocol the debugger uses to communicate with the target.
- After you make the option for the connection type, the **Connection Settings** changes to display configuration options specific for the hardware probe.

The topics in this chapter discuss the features and settings of the connections that interface the CodeWarrior debugger with the Kinetis device family.

- Changing Connection in IDE
- P&E ARM Multilink/Cyclone MAX/OSJTAG

Changing Connection in IDE

To change connection in the IDE, perform these steps.

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is of the pattern *projectname_1*, where ***projectname*** represents the name of the project.

NOTE NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the tree control and select the derivative or board you would like to use.

5. Click **Next**.

   The **Connections** page appears.

6. Select the desired connection.

NOTE You can select multiple connections by checking appropriate checkboxes in the **Connections** page.

7. Click **Finish**.

   The wizard creates a simulator project according to your specifications. You can access the project from the CodeWarrior Projects view in the **Workbench** window.

8. Build the new project. For more information, refer to the topic Building Projects.
9. Debug the new project. For more information, refer to the topic Debugging Projects.

P&E ARM Multilink/Cyclone MAX/OSJTAG

This section describes ARM P&E Connection options. The ARM P&E Connection setting permits a connection to ARM Freescale devices via P&E Multilink/Cyclone Pro/OSJTAG hardware interfaces. This connection mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor.

Connection Options

This topic describes all P&E ARM Multilink/Cyclone Pro/OSJTAG connection options, which are common to all P&E USB ARM Multilink or USB Multilink Universal, P&E Cyclone MAX Serial, P&E Cyclone MAX USB, P&E Cyclone MAX Ethernet, and P&E Open Source JTAG connections.

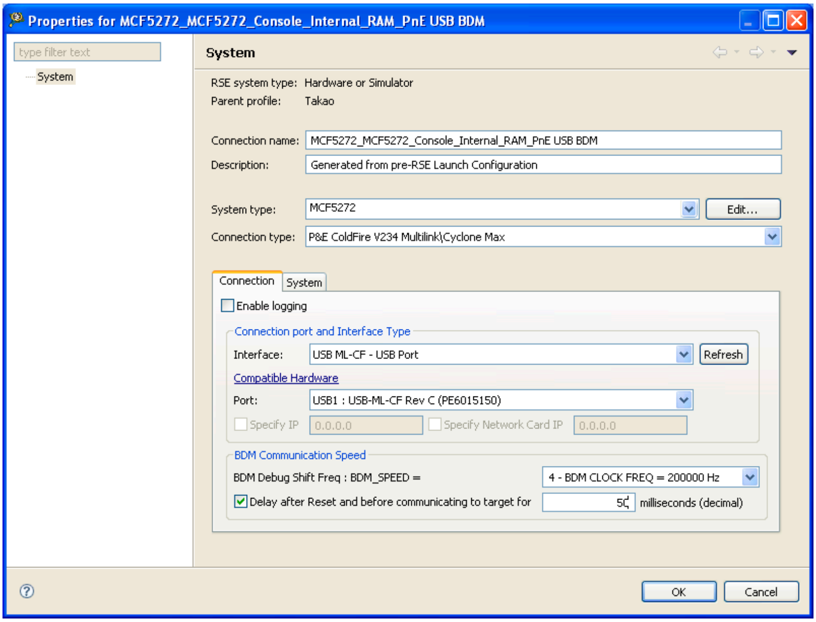
The options include:

- Changing P&E Connections Settings
- Connection Assistant
- Launch Configuration Settings
- Active Mode Menu Options
- Advanced Programming/Debug Options
- View Register Files Options
- P&E ARM Multilink/Cyclone MAX/OSJTAG Connection- Specific Options

Changing P&E Connections Settings

All connection settings for P&E hardware interfaces are configured in the Launch Configurations dialog box.

Figure 15.1 GDI — P&E ARM Multilink/Cyclone MAX/OSJTAG Connection



Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM

System

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0 ☐ Specify Network Card IP 0.0.0.0

BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

Table 15.1 describes the options for this view.

Table 15.1 Connection Parameter Options for P&E ARM Multilink/Cyclone MAX/OSJTAG

Option	Description
Interface	Use this option to select the interface type. Select a supported interface from the list box. The options are: • USB ARM Multilink - USB Port • USB Multilink Universal - USB Port **NOTE:** The USB Multilink Universal can conveniently support all Freescale architectures found in the current CodeWarrior 10 version • Cyclone MAX - Serial Port • Cyclone MAX - USB Port • Cyclone MAX - Ethernet Port **NOTE:** Click on the “Compatible Hardware” link to help you determine which P&E hardware is most suitable for your project. • OSJTAG
Port	This option selects the port over which debug communications is conducted. Select an available port from the list box. **NOTE**: If you are having issues trying to get a port to display, please click on the [FAQ #29] link for help.
Refresh	Click this button to have the workstation scan for a valid interface and port. Valid interfaces and ports appear in the `Interface` and `Port` list boxes.

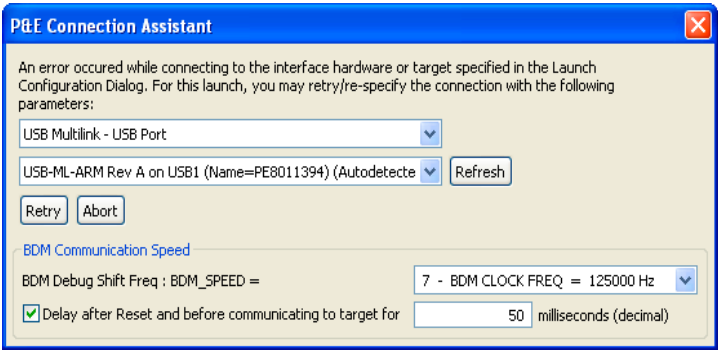
Option	Description
BDM Debug Shift Freq.	This option lets you to set the BDM shift clock speed of P&E's BDM interfaces. This integer value may be used to determine the speed of communications according to the following equations: Cyclone Max: (50000000/(2*N+5)) Hz USB-ML-ARM: (1000000/(N+1)) Hz USB Multilink Universal: (1000000/(N+1)) Hz The value n should be between 0 and 31. This shift clock takes effect after the commands in the top of the programming algorithm are executed so that these commands can increase the target frequency and allow a faster shift clock. This clock can't generally exceed a div 4 of the processor bus frequency.
Delay After Reset	Specifies a delay after the programmer resets the target that we check to see if the part has properly gone into background debug mode. This is useful if the target has a reset driver which hold the MCU in reset after the programmer releases the reset line. The n value is a delay in milliseconds.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug but the program cannot connect to the interface hardware specified in the Launch Configuration dialog box. To select the P&E ARM Multilink/Cyclone Max/OSJTAG as your debugger connection:

1. Choose the **P&E** device that you are using from the first drop-down menu and click **Refresh**. Refer to Figure 15.2.
2. Using the second drop-down menu, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 15.2 Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-ARM Rev A on USB1 (Name=PE8011394) (Autodetecte

BDM Debug Shift Freq : BDM_SPEED = 7 - BDM CLOCK FREQ = 125000 Hz

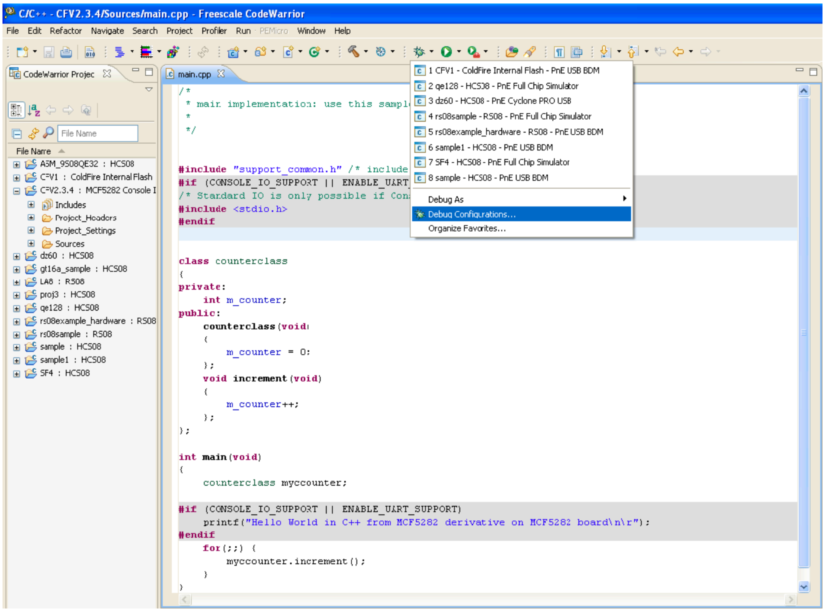
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

Launch Configuration Settings

To set the launch configurations for the debugger:

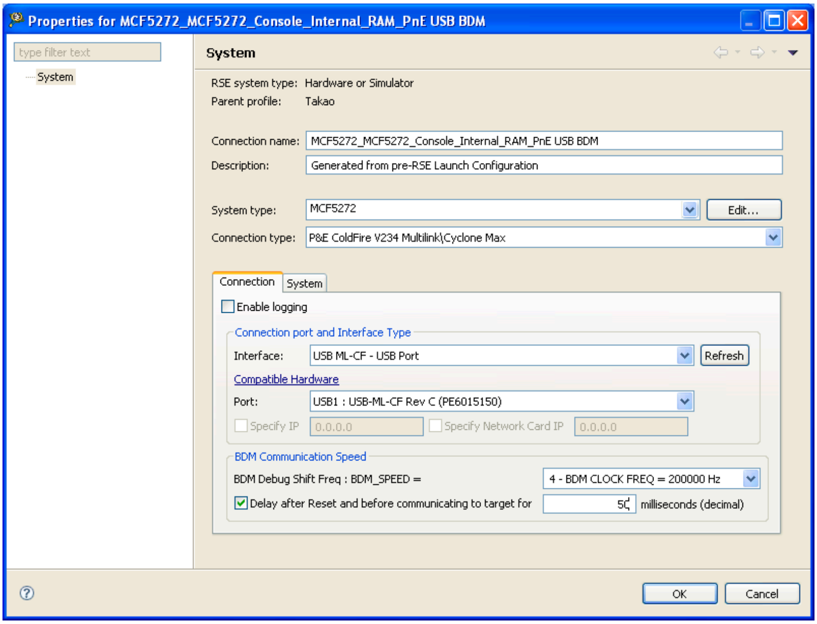
1. Find the debugger icon and click on the drop-down arrow to bring up the debugger menu. Refer to Figure 15.2.
2. Select **Debug Configurations**.
3. In the left column, select the project download type for which you would like to set the launch configurations. Refer to Figure 15.3
4. In the right column, set your configurations and click **Debug** to start the debugger.

Figure 15.3 Debugger Drop-down Menu


C/C++ - CFV2.3.4/Sources/main.cpp - Freescale CodeWarrior
File Edit Refactor Navigate Search Project Profiler Run PEMicro Window Help
CodeWarrior Projec
File Name
A3M_9S08QE32 : HCS08
CFV1 : ColdFire Internal Flash
CFV2.3.4 : MCF5282 Console I
Includes
Project_Headers
Project_Settings
Sources
dz60 : HCS08
gt16a_sample : HCS08
LA8 : RS08
proj3 : HCS08
qe128 : HCS08
rs08example_hardware : RS08
rs08sample : RS08
sample : HCS08
sample1 : HCS08
SF4 : HCS08
main.cpp
1 CFV1 - ColdFire Internal Flash - PnE USB BDM
2 qe128 - HCS08 - PnE Full Chip Simulator
3 dz60 - HCS08 - PnE Cyclone PRO USB
4 rs08sample - RS08 - PnE Full Chip Simulator
5 rs08example_hardware - RS08 - PnE USB BDM
6 sample1 - HCS08 - PnE USB BDM
7 SF4 - HCS08 - PnE Full Chip Simulator
8 sample - HCS08 - PnE USB BDM
Debug As
Debug Configurations...
Organize Favorites...

```
/*
 * main implementation: use this sampl
 *
 */


#include "support_common.h" /* include
#if (CONSOLE_IO_SUPPORT || ENABLE_UART
/* Standard IO is only possible if Con
#include <stdio.h>
#endif


class counterclass
{
private:
    int m_counter;
public:
    counterclass(void)
    {
        m_counter = 0;
    };
    void increment(void)
    {
        m_counter++;
    };
};

int main(void)
{
    counterclass myccounter;

#if (CONSOLE_IO_SUPPORT || ENABLE_UART_SUPPORT)
    printf("Hello World in C++ from MCF5282 derivative on MCF5282 board\n\r");
#endif
    for(;;) {
        myccounter.increment();
    }
}
```

Figure 15.4 Debugger Configuration Settings Dialog Box


Properties for MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
type filter text
System
System
Edit...
Connection
System
Refresh
Compatible Hardware
OK
Cancel

RSE system type: Hardware or Simulator
Parent profile: Takao

Connection name: MCF5272_MCF5272_Console_Internal_RAM_PnE USB BDM
Description: Generated from pre-RSE Launch Configuration

System type: MCF5272
Connection type: P&E ColdFire V234 Multilink\Cyclone Max

☐ Enable logging

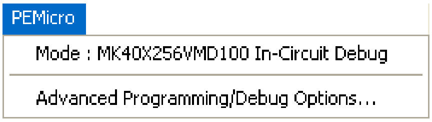
Connection port and Interface Type
Interface: USB ML-CF - USB Port
Port: USB1 : USB-ML-CF Rev C (PE6015150)
☐ Specify IP 0.0.0.0
☐ Specify Network Card IP 0.0.0.0

BDM Communication Speed
BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 200000 Hz
☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

Active Mode Menu Options

When the microprocessor is connected, the active mode menu shows the name of the microprocessor and gives you the access to the Advanced Programming/Debug Options. When the microprocessor is not connected, the menu is not available.

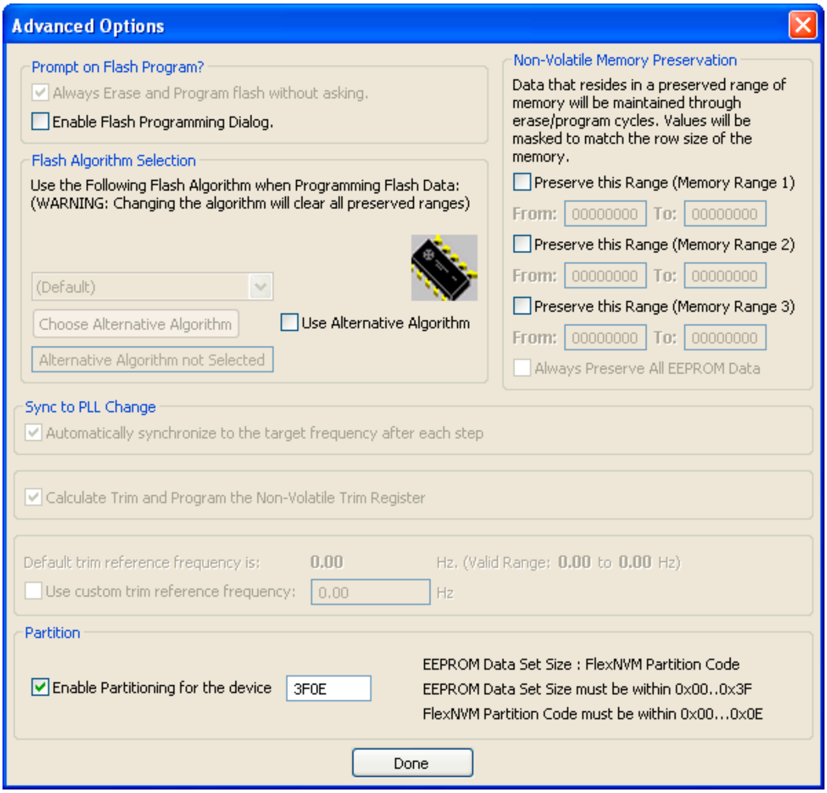
Figure 15.5 Additional Connection Menu Options


PEMicro
Mode : MK40X256VMD100 In-Circuit Debug
Advanced Programming/Debug Options...

Advanced Programming/Debug Options

The Advanced Programming/Debug Options menu option takes you to the Advanced Options dialog box, where you can configure the software settings for the flash programming procedure.

Figure 15.6 Advanced Options Dialog Box



Advanced Options

Prompt on Flash Program?

☑ Always Erase and Program flash without asking.

☐ Enable Flash Programming Dialog.

Flash Algorithm Selection

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

Choose Alternative Algorithm

☐ Use Alternative Algorithm

Alternative Algorithm not Selected

Non-Volatile Memory Preservation

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)

From: 00000000 To: 00000000

☐ Preserve this Range (Memory Range 2)

From: 00000000 To: 00000000

☐ Preserve this Range (Memory Range 3)

From: 00000000 To: 00000000

☐ Always Preserve All EEPROM Data

Sync to PLL Change

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 0.00 Hz. (Valid Range: 0.00 to 0.00 Hz)

☐ Use custom trim reference frequency: 0.00 Hz

Partition

☑ Enable Partitioning for the device 3F0E

EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Enable Flash Programming Dialog

Setting the Enable Flash Programming dialog box lets you view the steps taken by the Flash Programmer.

Non-Volatile Memory Preservation

You have the option of preserving up to three independent ranges of non-volatile memory (on devices with EEPROM, the EEPROM array may optionally be preserved as well). Ranges that are designated as “preserved” are read before an erase, and reprogrammed immediately afterwards, thereby preserving the data in these ranges. Any attempt to program data into a preserved range is ignored. When entering an address into the preserved range field (hexadecimal input is expected), the values are masked according to the row size of the device. This ensures that the reprogramming of preserved data does not cause any conditions that disturb programming.

Alternative Algorithm Functionality

Once you create a project for a specific Kinetis microprocessor, the debugger specifies a default algorithm to use during all Flash programming operations. The debugger uses this algorithm for nearly all programming requirements. The default algorithm can be found in the <CW_Install>/MCU/bin/plugins/support/ARM/gdi/P&E directory

However, you can override the default algorithm via the Alternative Algorithm function, located in the Advanced Programming/Debug Options menu. This feature can be used to select a custom programming algorithm, or select another one of P&E's many programming algorithms for use with a specific project.

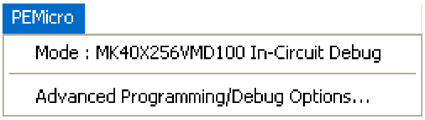
TIP Selecting a wrong programming algorithm may damage you device, lead to under/over programming situations, or simply not program portions of the project file.

Therefore it is recommended to use the default algorithm unless there is a compelling reason to do otherwise.

Use these steps to override the default algorithm:

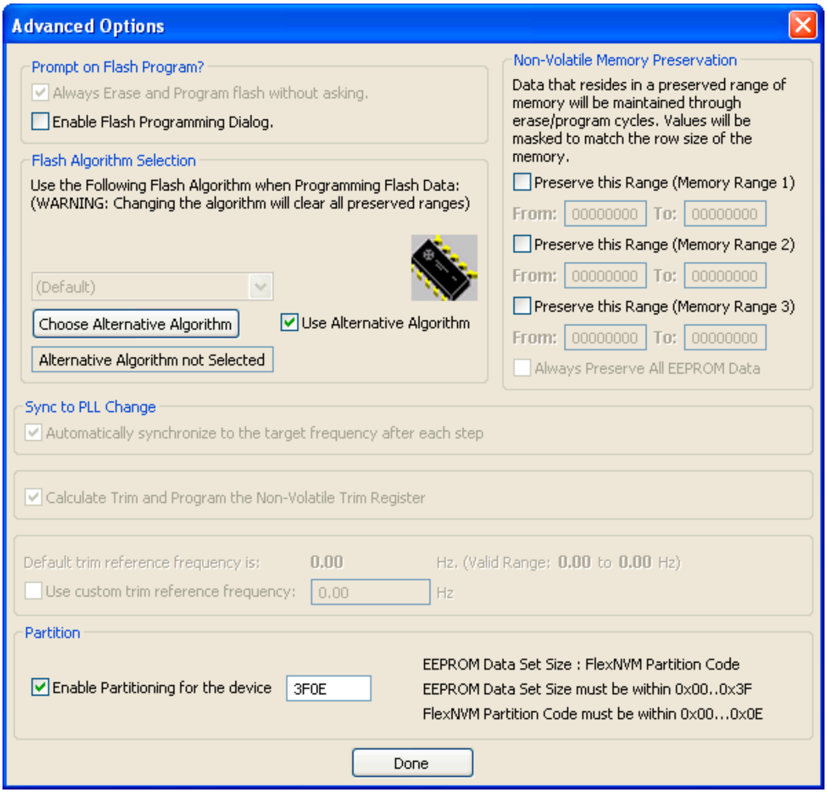
1. Select the Advanced Programming/Debug Options selection from the PEMicro menu.

Figure 15.7 Advanced Programming/Debug Options Menu Selection


PEMicro
Mode : MK40X256VMD100 In-Circuit Debug
Advanced Programming/Debug Options...

2. Check the Use Alternative Algorithm checkbox.

Figure 15.8 Advanced Options - Alternative Algorithm Checkbox



Advanced Options

Prompt on Flash Program?

☑ Always Erase and Program flash without asking.

☐ Enable Flash Programming Dialog.

Flash Algorithm Selection

Use the Following Flash Algorithm when Programming Flash Data:
(WARNING: Changing the algorithm will clear all preserved ranges)

(Default)

Choose Alternative Algorithm

☑ Use Alternative Algorithm

Alternative Algorithm not Selected

Non-Volatile Memory Preservation

Data that resides in a preserved range of memory will be maintained through erase/program cycles. Values will be masked to match the row size of the memory.

☐ Preserve this Range (Memory Range 1)

From: 00000000 To: 00000000

☐ Preserve this Range (Memory Range 2)

From: 00000000 To: 00000000

☐ Preserve this Range (Memory Range 3)

From: 00000000 To: 00000000

☐ Always Preserve All EEPROM Data

Sync to PLL Change

☑ Automatically synchronize to the target frequency after each step

☑ Calculate Trim and Program the Non-Volatile Trim Register

Default trim reference frequency is: 0.00 Hz. (Valid Range: 0.00 to 0.00 Hz)

☐ Use custom trim reference frequency: 0.00 Hz

Partition

☑ Enable Partitioning for the device 3F0E

EEPROM Data Set Size : FlexNVM Partition Code
EEPROM Data Set Size must be within 0x00..0x3F
FlexNVM Partition Code must be within 0x00...0x0E

Done

3. Click the Choose Alternative Algorithm button, which lets you browse for an alternative algorithm.

4. Once you select the alternative algorithm, the name of the algorithm along with its full path appears in the text field below the Choose Alternative Algorithm button.

At this point, the current project performs all future Flash programming operations using the alternative algorithm. You may revert to the default algorithm at any time by clearing the Use Alternative Algorithm checkbox.

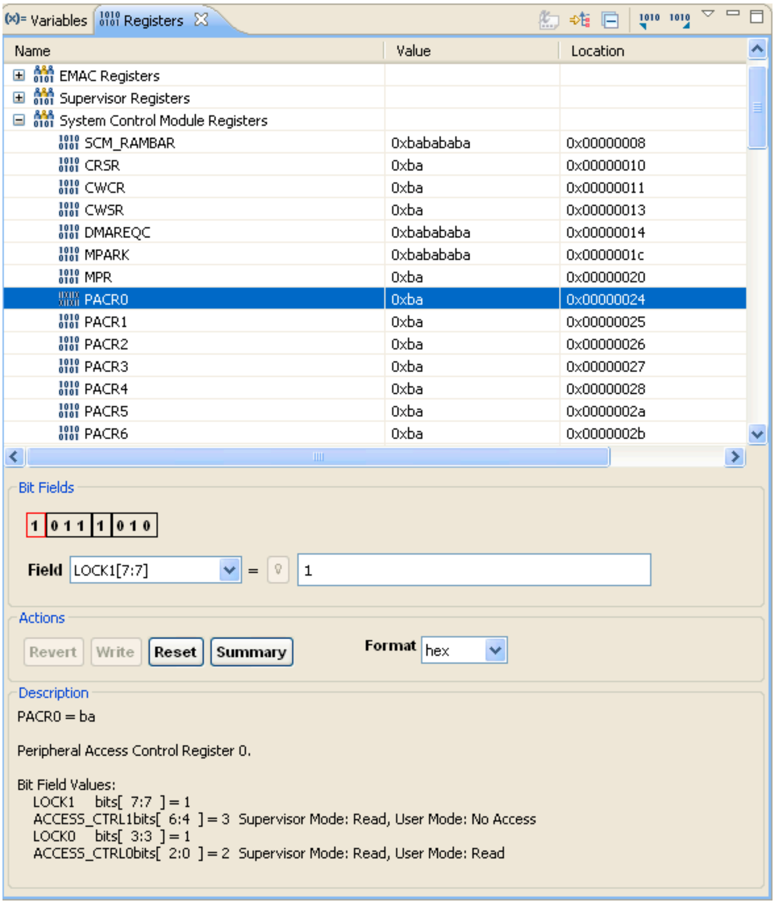
Enable Partitioning

Setting the Enable Partitioning checkbox enables you to partition your Kinetis chip. To setup partitioning, you must provide two bytes in hexadecimal format. The first byte represents the EEPROM Data Set Size, which determines the amount of FlexRAM used in each of the available EEPROM subsystems. This byte must be in the range of 0x00 to 0x3F. The second byte represents the FlexNVM Partition Code, which specifies how to split the FlexNVM block between data flash and EEPROM backup memory supporting EEPROM functions. This must be in the range of 0x00 to 0x0E. Please refer to the reference manual specific to the device you are working with for more information on how to specify the EEPROM Data Set Size and FlexNVM Partition Code.

View Register Files Options

The **Register Files** tab in the debugger gives you option of viewing and editing the register files. If register files are available for the device that you have chosen, the **Registers** tab in the debugger (see Figure 15.9) is populated.

Figure 15.9 Debug Register File Tab



Name	Value	Location
⊞ EMAC Registers		
⊞ Supervisor Registers		
⊟ System Control Module Registers		
SCM_RAMBAR	0xbabababa	0x00000008
CRSR	0xba	0x00000010
CWCR	0xba	0x00000011
CWSR	0xba	0x00000013
DMAREQC	0xbabababa	0x00000014
MPARK	0xbabababa	0x0000001c
MPR	0xba	0x00000020
PACR0	0xba	0x00000024
PACR1	0xba	0x00000025
PACR2	0xba	0x00000026
PACR3	0xba	0x00000027
PACR4	0xba	0x00000028
PACR5	0xba	0x0000002a
PACR6	0xba	0x0000002b

1	0 1 1	1	0 1 0

Field LOCK1[7:7] = 1

Format hex

```
PACR0 = ba

Peripheral Access Control Register 0.

Bit Field Values:
   LOCK1    bits[  7:7  ] = 1
   ACCESS_CTRL1bits[  6:4  ] = 3  Supervisor Mode: Read, User Mode: No Access
   LOCK0    bits[  3:3  ] = 1
   ACCESS_CTRL0bits[  2:0  ] = 2  Supervisor Mode: Read, User Mode: Read
```

To view the **Register Files** of the device that you have chosen:

1. Find the debugger icon and click it to enter debug mode and open the debugging window.
2. Select the **Registers** tab on the right side of the debugging window, or select the **Window menu -> Show View -> Registers** to open the **Register** window.
3. Expand a module by clicking on the plus/minus button to view the registers within the module
4. Select a desired register to view its bit fields and bit descriptions in the window below.

In the **Registers** tab, all of the available modules are listed, and under each module all of its registers are displayed with their current values. Selecting a register brings up the Bit field, Actions box, and Description box. In the Bit field, you can view the bits in binary format. The Actions box is used when a bit needs to be modified. You can revert changes, write a new value, reset all of the bits, and view a summary of the register. You can also change the format of the value written in the bit field. The Description box displays the values and significance of each bit in the register. When a bit is modified, the description will change.

You can modify each bit by selecting it in the drop-down box under the Bit field, or by clicking the bit on the Bit field. Note that bits that are read-only will not allow you to modify the bit values. A new value can be written into the edit box, or you can click the light bulb button next to the edit box to view all of the options, and then double-click the changes.

P&E ARM Multilink/Cyclone MAX/OSJTAG Connection- Specific Options

This topic describes the connection-specific options. The connections include:

- P&E USB ARM Multilink or USB Multilink Universal
- P&E Cyclone MAX Serial
- P&E Cyclone MAX USB
- P&E Cyclone MAX Ethernet
- P&E Open Source JTAG

P&E USB ARM Multilink or USB Multilink Universal

The **P&E USB ARM Multilink** or **USB Multilink Universal Connection** setting permits a connection to USB ARM Multilink devices. P&E USB ARM Multilink mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources. The Multilink connection option can also be used to connect a USB Multilink Universal to your interface.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E USB ARM Multilink** or **USB Multilink Universal** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Kinetis** tree control and select the derivative or board you would like to use. For example, select **Kinetis > MK40 Family > MK40N512VMD100**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E USB ARM Multilink** checkbox. Refer to Figure 15.10.
7. Click **Finish**.

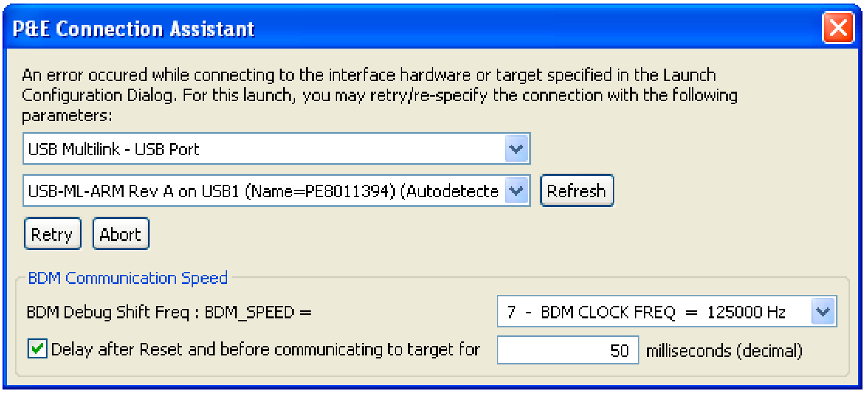
The wizard creates a project for the Kinetis ARM architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E USB ARM Multilink** or **USB Multilink Universal** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 15.10.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details of each setting.
4. Click the **Retry** button.

Figure 15.10 Kinetis ARM Connection Assistant Interface Selected



P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

USB-ML-ARM Rev A on USB1 (Name=PE8011394) (Autodetecte

BDM Communication Speed

BDM Debug Shift Freq : BDM_SPEED = 7 - BDM CLOCK FREQ = 125000 Hz

☑ Delay after Reset and before communicating to target for 50 milliseconds (decimal)

P&E Cyclone MAX Serial

The P&E Cyclone MAX Serial Connection setting permits a connection to Cyclone MAX Serial devices. P&E Cyclone MAX Serial mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Serial** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Kinetis** tree control and select the derivative or board you would like to use. For example, select **Kinetis > MK40 Family > MK40N512VMD100**.
5. Click **Next**.

   The **Connections** page appears.
6. Check the **P&E Cyclone MAX Serial** checkbox. If P&E Cyclone MAX Serial is not available, then select **P&E USB ARM Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**. Refer to Changing P&E Connections Settings for information on how to change the settings.
7. Click **Finish**.

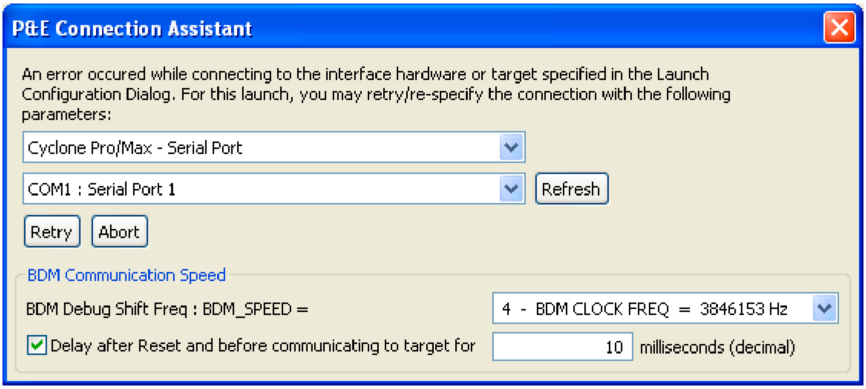
The wizard creates a project for the Kinetis ARM architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E Cyclone MAX Serial** as your debugger connection:

1. Select **Cyclone Pro/Max – Serial Port** from the first drop-down list and click **Refresh**. Refer to Figure 15.11.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 15.11 Kinetis ARM Connection Assistant Interface Selected


P&E Connection Assistant

An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Serial Port

COM1 : Serial Port 1

BDM Communication Speed

BDM Debug Shift Freq : BDM_SPEED = 4 - BDM CLOCK FREQ = 3846153 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX USB

The P&E Cyclone MAX USB Connection setting permits a connection to Cyclone MAX USB devices. P&E Cyclone MAX USB mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

NOTE In the event that the USB drivers do not install when using the Linux operating system, users may have to manually install P&E drivers to use USB-enabled devices. Please refer to the Readme.pdf found within the `[CodewarriorInstallDir]\Drivers\pemicro` for more details.

To select **P&E Cyclone MAX USB** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Kinetis** tree control and select the derivative or board you would like to use. For example, select **Kinetis > MK40 Family > MK40N512VMD100**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Cyclone MAX USB** checkbox. If P&E Cyclone MAX USB is not available, then select **P&E USB ARM Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**. Refer to Changing P&E Connections Settings for information on how to change the settings.
7. Click **Finish**.

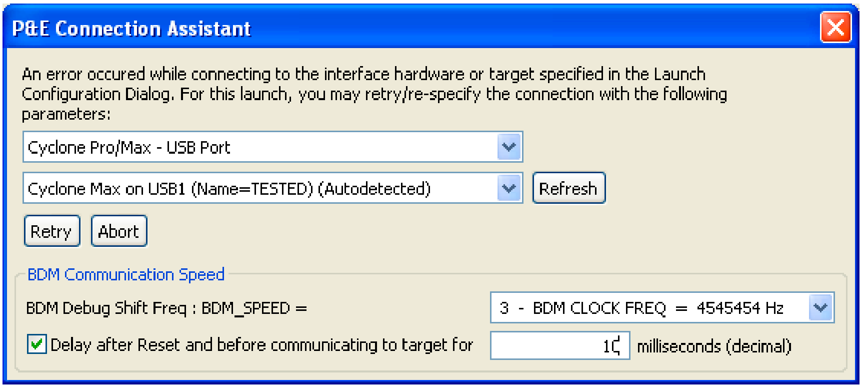
The wizard creates a project for the Kinetis ARM architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E Cyclone MAX USB** as your debugger connection:

1. Select **Cyclone Pro/Max – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 15.12.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details of each setting.
4. Click the **Retry** button.

Figure 15.12 Kinetis ARM Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - USB Port

Cyclone Max on USB1 (Name=TESTED) (Autodetected)

BDM Debug Shift Freq : BDM_SPEED = 3 - BDM CLOCK FREQ = 4545454 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Cyclone MAX Ethernet

The P&E Cyclone MAX Ethernet Connection setting permits a connection to Cyclone MAX TCP-IP devices. P&E Cyclone MAX Ethernet mode lets you debug code, as the firmware is fully resident in the Flash of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Cyclone MAX Ethernet** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Kinetis** tree control and select the derivative or board you would like to use. For example, select **Kinetis > MK40 Family > MK40N512VMD100**.
5. Click **Next**.

   The **Connections** page appears.

6. heck the **P&E Cyclone MAX Ethernet** checkbox. If P&E Cyclone MAX Ethernet is not available, then select **P&E USB ARM Multilink** temporarily, and later you will have to change the P&E connection settings within **Debug Configurations**. Refer to Changing P&E Connections Settings for information on how to change the settings.
7. Click **Finish**.

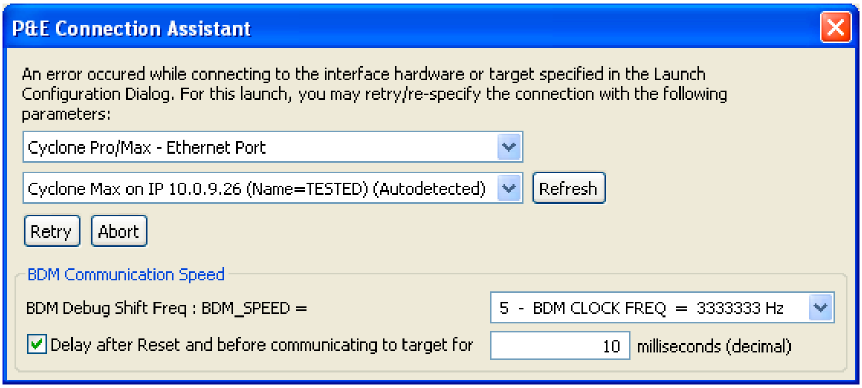
The wizard creates a project for the Kinetis ARM architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select the **P&E Cyclone MAX Ethernet** as your debugger connection:

1. Select **Cyclone Pro/Max – Ethernet Port** from the first drop-down list and click **Refresh**. Refer to Figure 15.13.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details of each setting.
4. Click the **Retry** button.

Figure 15.13 Kinetis ARM Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

Cyclone Pro/Max - Ethernet Port

Cyclone Max on IP 10.0.9.26 (Name=TESTED) (Autodetected)

BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 3333333 Hz

☑ Delay after Reset and before communicating to target for 10 milliseconds (decimal)

P&E Open Source JTAG

Freescale supplies certain development boards with an integrated debug circuit based on Open Source JTAG. This allows the development board to be debugged from the PC via the USB bus without requiring external debug hardware, such as the Cyclone Max or USB ARM Multilink. The development board also derives its power from the USB Bus.

The Open Source JTAG circuit design (OSJTAG-JM60) is an open source, community driven design. It has been published on Freescale's website, and full documentation can be found in the Community Forums.

Integration with CodeWarrior is handled via the "P&E Open Source JTAG" connection. P&E has integrated the Open Source JTAG support into the same connection that supports both the USB ColdFire Multilink and the Cyclone Max. All of the dialogs that affect operation of these hardware interfaces function in the same manner when using OSJTAG (albeit at a lower data rate).

The P&E Open Source JTAG Connection setting permits a connection to Open Source JTAG devices. Open Source JTAG mode lets you debug code, as the firmware is fully resident in the Flash or RAM of the microprocessor. The operation of all modules fully reflects the actual operation of the onboard resources.

To select **P&E Open Source JTAG** as the debugger connection:

1. Select **Project > Change Device/Connection** from the IDE menu bar.

   The **Device/Connection Change** wizard appears.

2. Specify a name for the project, in the **New Project Name** text box. By default, it is the existing project name.

NOTE Clear the **Use default location** checkbox and click **Browse** to specify a different location for the new project. By default, the **Use default location** checkbox is checked.

3. Click **Next**.

   The **Devices** page appears.

4. Expand the **Kinetis** tree control and select the derivative or board you would like to use. For example, select **Kinetis > MK40 Family > MK40N512VMD100**.
5. Click **Next**.

   The **Connections** page appears.

6. Check the **P&E Open Source JTAG** checkbox. Refer to Figure 15.14.
7. Click **Finish**.

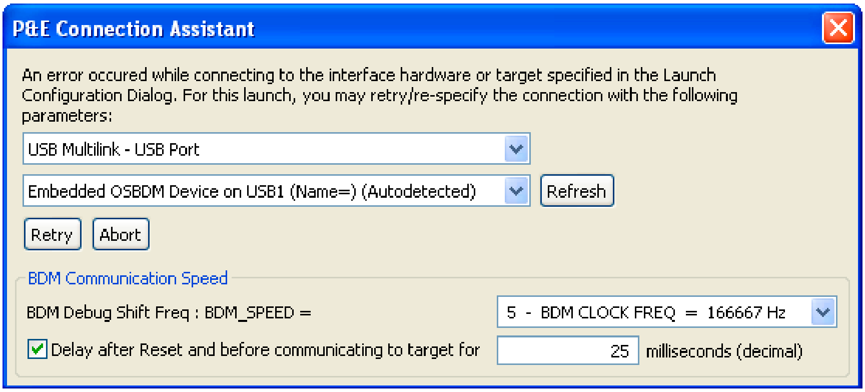
The wizard creates a project for the Kinetis ARM architecture according to your specifications. You can access the project from the CodeWarrior projects view in the **Workbench** window.

Connection Assistant

The P&E Connection Assistant is displayed when you attempt to debug and the program cannot connect to the interface hardware specified in the **Launch Configuration** dialog box. To select **P&E Open Source JTAG** as your debugger connection:

1. Select **USB Multilink – USB Port** from the first drop-down list and click **Refresh**. Refer to Figure 15.14.
2. Using the second drop-down list, select the port on which the interface is connected.
3. Use the **BDM Communication Speed** panel to configure the shift frequency and delay. Refer to Table 15.1 for more details regarding each setting.
4. Click the **Retry** button.

Figure 15.14 Kinetis ARM Connection Assistant Interface Selected



An error occured while connecting to the interface hardware or target specified in the Launch Configuration Dialog. For this launch, you may retry/re-specify the connection with the following parameters:

USB Multilink - USB Port

Embedded OSBDM Device on USB1 (Name=) (Autodetected)

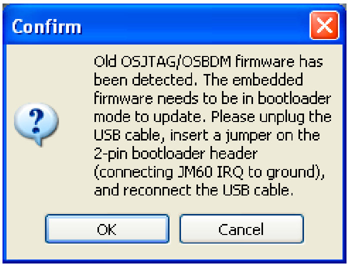
BDM Debug Shift Freq : BDM_SPEED = 5 - BDM CLOCK FREQ = 166667 Hz

☑ Delay after Reset and before communicating to target for 25 milliseconds (decimal)

OSJTAG Firmware Update

All Codewarrior IDE’s version 10.1 and higher have an automatic firmware update mechanism for built-in OSJTAG hardware interfaces. Whenever an OSJTAG-integrated hardware interface is plugged into a USB port and Codewarrior attempts to contact the device, it will automatically check to see if the device has the latest OSJTAG firmware version. If the firmware on the device is older than the one found within the Codewarrior package, then a dialog box will indicate that a firmware update is necessary.

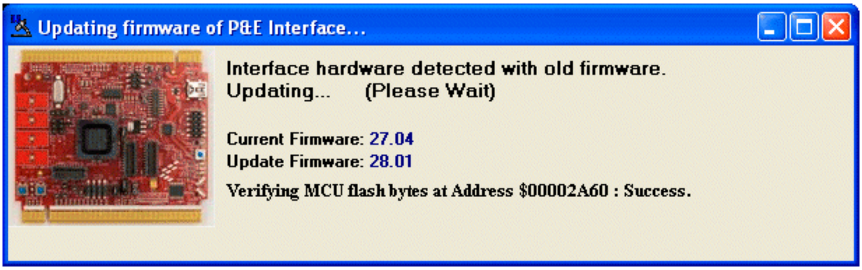
Figure 15.15 Old OSJTAG Firmware Detected



Old OSJTAG/OSBDM firmware has been detected. The embedded firmware needs to be in bootloader mode to update. Please unplug the USB cable, insert a jumper on the 2-pin bootloader header (connecting JM60 IRQ to ground), and reconnect the USB cable.

To update the firmware, the OSJTAG device must enter Bootloader mode. To do so the USB cable must be disconnected from the device and the OSJTAG-JM60 IRQ pin must be connected to ground usually done by using a 2-pin female jumper. Use the OSJTAG device schematics to find the IRQ pin. Once the IRQ pin is grounded, connect the USB cable to the OSJTAG device and click **OK**. If done correctly, the automatic firmware update will occur.

Figure 15.16 OSJTAG Firmware Updating

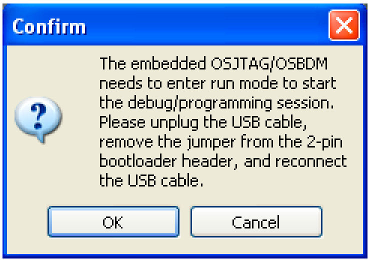


Interface hardware detected with old firmware.
Updating... (Please Wait)

Current Firmware: 27.04
Update Firmware: 28.01
Verifying MCU flash bytes at Address $00002A60 : Success.

When the firmware is done updating, a dialog box will indicate that the OSJTAG device must exit Bootloader mode and enter into Run mode.

Figure 15.17 Start OSJTAG Run Mode



The embedded OSJTAG/OSBDM needs to enter run mode to start the debug/programming session. Please unplug the USB cable, remove the jumper from the 2-pin bootloader header, and reconnect the USB cable.

To enter Run Mode, the user must diconnect the USB cable from the OSJTAG device and the 2-pin female jumper on the IRQ pin must be removed. Next, reconnect the USB cable and the device will be in Run Mode. Click **OK** and Codewarrior will move onto programming or running the code.

The Codewarrior IDE layout will have the latest OSJTAG firmware. If for any reason you experience difficulty performing OSJTAG firmware update, please visit `www.pemicro.com/osbdm` and use the Firmware Update Utility to force an update via an external USB-ML-12 hardware interface.

16

Common Connection Features

This chapter explains how to use the CodeWarrior hardware tools. Use these tools for board bring-up, test, and analysis.

The topics in this chapter are:

- Working with Flash Programmer
- Quick Access to Target Tasks
- Flash Programmer Tutorials
- Working with Hardware Diagnostics Window
- Manipulating Target Memory

Working with Flash Programmer

The CodeWarrior Flash Programmer can program the flash memory of the target board with code from any CodeWarrior IDE project or any individual files. The Flash Programmer (FP) feature is a target task that lets you run a series of actions on a flash: internal or present on a board (NOR, NAND, etc.). The supported operations are:

- **Program** — Program an image in flash (address restrictions may apply).
- **Erase** — Erase the flash at a sector/block level, and all flash if possible.
- **Blank Check** — At sector level and when possible, all flash.
- **Verify** — Verify the programmed image (address restrictions may apply).
- **Checksum** — Checksum of a written image (file on target, file on host, memory range on target, and all flash).

The CodeWarrior Flash Programmer lets you program the flash memory of any of the supported target boards, from within the IDE. You can do this using either the pre-defined tasks provided with the CodeWarrior installation, or create your own specialized tasks. Each of these options is described in the following topics:

- Use Pre-Defined Programming Task
- Create Flash Programmer Task
- Flash Programmer for MCF5441X

Use Pre-Defined Programming Task

To use a pre-defined flash programming task, you first import its `*.xml` file into the **Target Tasks** view. CodeWarrior for Microcontrollers provides default flash configuration files for a wide variety of supported target boards. The pre-defined task files are in the following directories:

`<CWInstallDir>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/HC08`

`<CWInstallDir>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/ColdFire`

After you have imported the task, it appears in the **Target Tasks** view, where you can execute it.

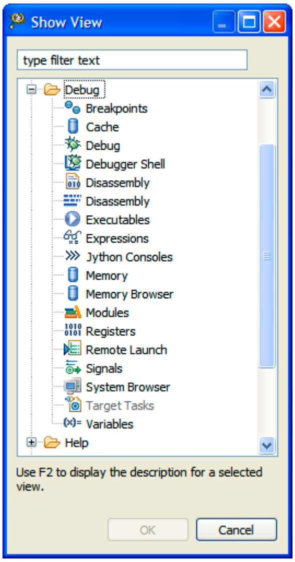
To import a pre-defined Flash Programmer task, perform these steps.

1. From the CodeWarrior menu bar, select **Window > Show View > Other**.

   The **Show View** dialog box appears.

2. Expand the **Debug** tree control and select **Target Tasks**. Refer Figure 16.1.

Figure 16.1 Show View Dialog Box

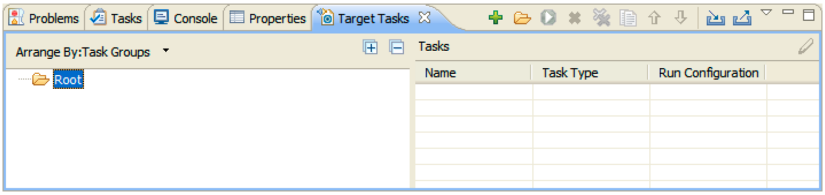

Show View
type filter text
Debug
Breakpoints
Cache
Debug
Debugger Shell
Disassembly
Disassembly
Executables
Expressions
Jython Consoles
Memory
Memory Browser
Modules
Registers
Remote Launch
Signals
System Browser
Target Tasks
Variables
Help
OK
Cancel

Use F2 to display the description for a selected view.

3. Click **OK**.

   The **Target Tasks** view appears. Refer Figure 16.2.

Figure 16.2 Target Tasks View


Problems
Tasks
Console
Properties
Target Tasks
Arrange By:Task Groups
Root
Tasks
Name
Task Type
Run Configuration

4. Right-click in the **Target Tasks** view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

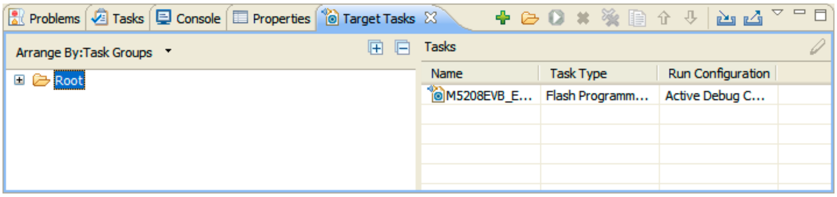
   The **Open** dialog box appears.

5. Navigate to the pre-defined tasks folder at `<CW MCU install>\MCU\bin\plugins\support\TargetTask\Flash_Programmer\` and select the desired `.xml` file for your hardware target. For example, select `MCF5213_INTFLASH.xml` from the `ColdFire` folder.

6. Click **Open**.

   The selected task appears in the **Target Tasks** view (Figure 16.3).

Figure 16.3 Pre-defined Task in Target Tasks View


Problems
Tasks
Console
Properties
Target Tasks
Arrange By:Task Groups
Root
Tasks

Name	Task Type	Run Configuration
M5208EVB_E...	Flash Programm...	Active Debug C...

7. Right-click on the task’s name and select **Execute**.

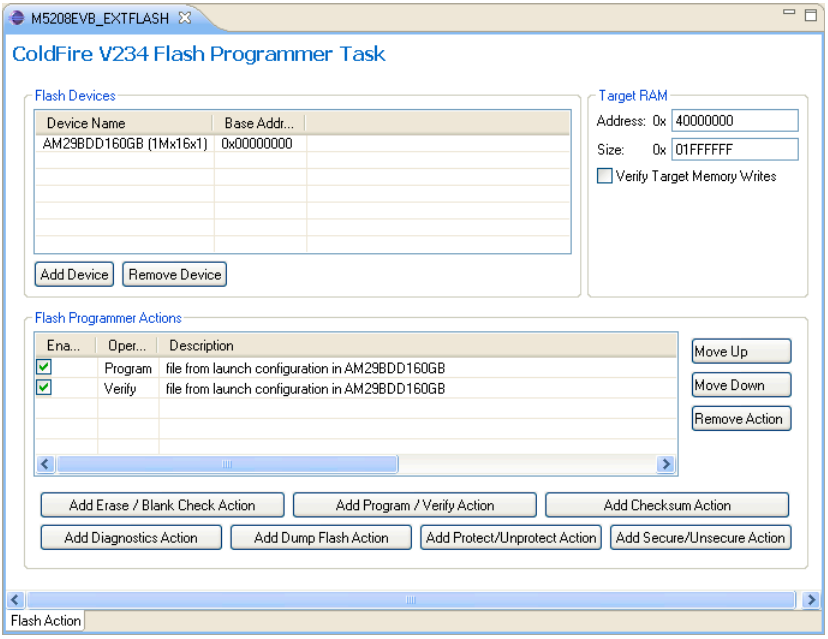
   The task’s Flash Programmer actions execute in sequence. First, they erase the hardware target’s flash memory.

NOTE When a predefined flash programmer task is imported, its **Run Configuration** is set as `Active Debug Context`. If the task is imported and there is no active debug session, then the ▷ icon will be disabled as in Figure 16.3. Associate the selected target task to a different Run Configuration to enable the ▶ icon.

8. Double-click on the task’s name, to examine the task’s stored Flash Programmer actions.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the actions in the **Flash Programmer Actions** group (Figure 16.4).

Figure 16.4 *<target>* Flash Programmer Task Editor Window Displaying Stored Actions


M5208EVB_EXTFLASH
Add Device
Remove Device
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

ColdFire V234 Flash Programmer Task

Flash Devices

Device Name	Base Addr...
AM29BDD160GB (1Mx16x1)	0x00000000

Target RAM

Address: 0x 40000000

Size: 0x 01FFFFFF

☐ Verify Target Memory Writes

Flash Programmer Actions

Ena...	Oper...	Description
☑	Program	file from launch configuration in AM29BDD160GB
☑	Verify	file from launch configuration in AM29BDD160GB

If you are working with special hardware that require a different sequence of Flash Programmer actions, you can create your own target tasks.

NOTE For more information on creating Flash Programmer target tasks, refer to the Create Flash Programmer Task topic.

For more information on the various options available in the ***<target>* Flash Programmer Task** editor window, refer to the *Freescale Eclipse Extensions Guide.*

Create Flash Programmer Task

In the Eclipse IDE, the Flash Programmer runs like a target task.

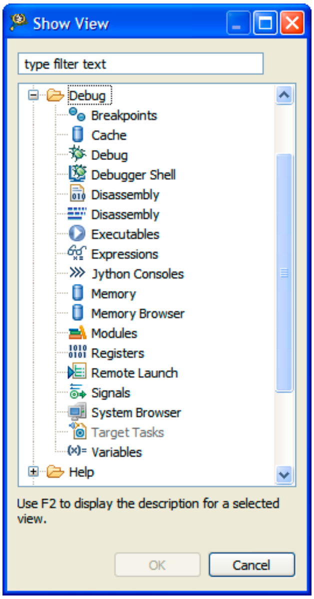
To create a Flash Programmer target task, perform these steps.

1. From the CodeWarrior main menu bar, select **Window > Show View > Other**.

   The **Show View** dialog box appears.

2. Expand the **Debug** tree control and select **Target Tasks**. Refer Figure 16.5.

Figure 16.5 Show View Dialog Box

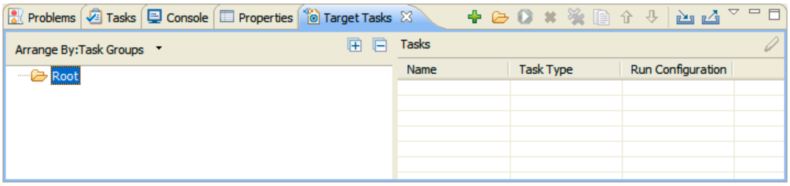

Show View
type filter text
Debug
Breakpoints
Cache
Debug
Debugger Shell
Disassembly
Disassembly
Executables
Expressions
Jython Consoles
Memory
Memory Browser
Modules
Registers
Remote Launch
Signals
System Browser
Target Tasks
Variables
Help
OK
Cancel

Use F2 to display the description for a selected view.

3. Click **OK**.

   The **Target Tasks** view appears. Refer Figure 16.6.

Figure 16.6 Target Tasks View


Problems
Tasks
Console
Properties
Target Tasks
Arrange By:Task Groups
Root
Tasks
Name
Task Type
Run Configuration

4. Click ✚ from the **Target Tasks** view toolbar to create a new target task.

   The **Create New Target Task** wizard appears.

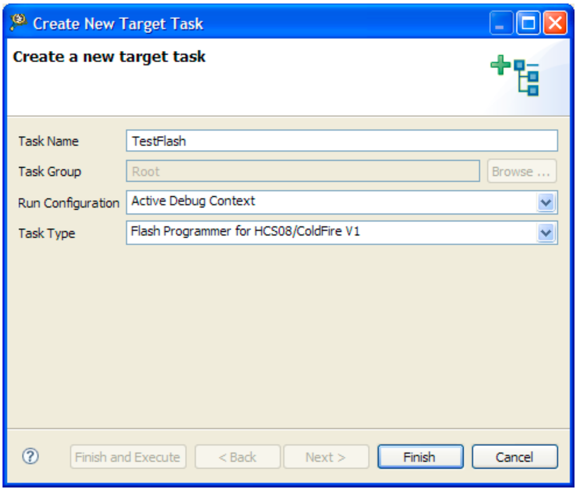
5. In the **Task Name** text box, enter the name of the target task.

6. From the **Run Configuration** drop-down list, select a configuration.

NOTE Select **Active Debug Context** from the **Run Configuration** drop-down list, if you want to use Flash Programmer over an active debug session, else select any of the specified debug context from the list.

7. From the **Task Type** drop-down list, select the appropriate **Flash Programmer**. Refer Figure 16.7. If necessary, select `Flash Programmer for HCS08/ColdFire V1` from the drop-down list.

Figure 16.7 Create New Target Task Wizard



Create New Target Task
Create a new target task

Task Name: TestFlash
Task Group: Root
Run Configuration: Active Debug Context
Task Type: Flash Programmer for HCS08/ColdFire V1

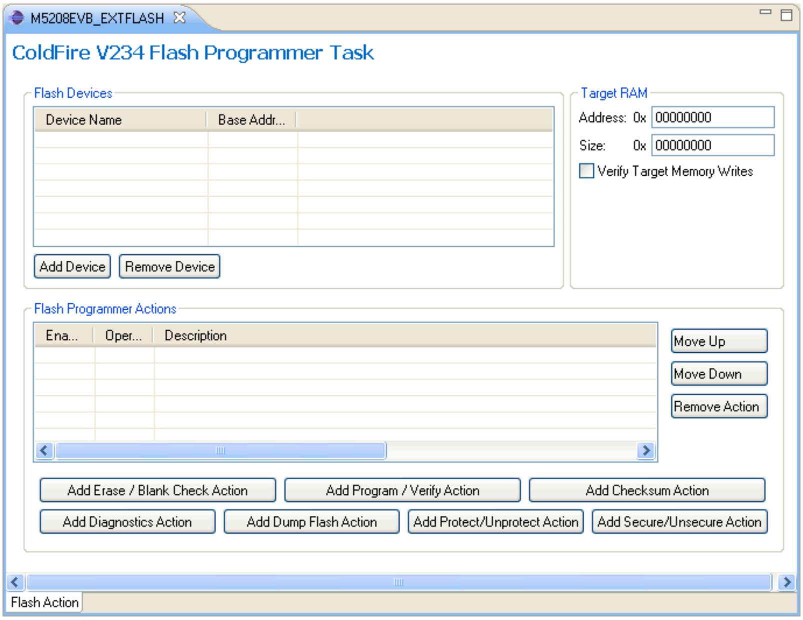
Browse ...
Finish and Execute
< Back
Next >
Finish
Cancel

8. Click **Finish**.

   The ***<target>* Flash Programmer Task** editor window appears as seen in Figure 16.8.

NOTE The ***<target>* Flash Programmer Task** editor window has groups to define flash devices, Flash Programmer actions, and target RAM settings.

Figure 16.8 *<target>* Flash Programmer Task Editor Window


M5208EVB_EXTFLASH
ColdFire V234 Flash Programmer Task
Flash Devices
Device Name
Base Addr...
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Ena...
Oper...
Description
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

Address: 0x 00000000
Size: 0x 00000000
☐ Verify Target Memory Writes

9. Click **Add Device** from the **Flash Devices** group to add a new hardware device.

   The **Add Device** dialog box appears with a list of supported devices.

   a. Select the specific device from the list.

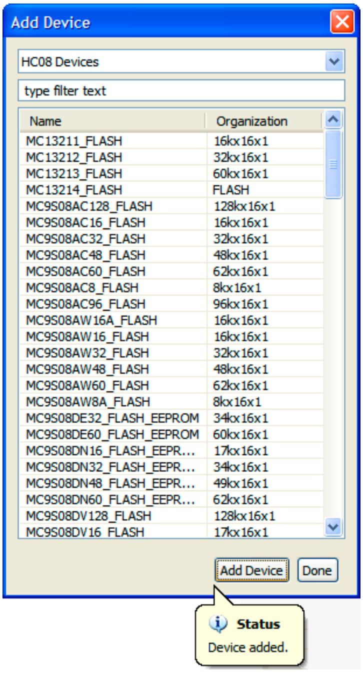
   b. Change the device's memory organization (if required).

NOTE To change the device memory organization, click the adjacent value in the **Organization** column. Click the ▾ icon, and select the required organization.

   c. Click **Add Device**.

   You get a popup with a status that the device is added.

Figure 16.9 Add Device Dialog Box - Popup with Status



Name	Organization
MC13211_FLASH	16kx16x1
MC13212_FLASH	32kx16x1
MC13213_FLASH	60kx16x1
MC13214_FLASH	FLASH
MC9S08AC128_FLASH	128kx16x1
MC9S08AC16_FLASH	16kx16x1
MC9S08AC32_FLASH	32kx16x1
MC9S08AC48_FLASH	48kx16x1
MC9S08AC60_FLASH	62kx16x1
MC9S08AC8_FLASH	8kx16x1
MC9S08AC96_FLASH	96kx16x1
MC9S08AW16A_FLASH	16kx16x1
MC9S08AW16_FLASH	16kx16x1
MC9S08AW32_FLASH	32kx16x1
MC9S08AW48_FLASH	48kx16x1
MC9S08AW60_FLASH	62kx16x1
MC9S08AW8A_FLASH	8kx16x1
MC9S08DE32_FLASH_EEPROM	34kx16x1
MC9S08DE60_FLASH_EEPROM	60kx16x1
MC9S08DN16_FLASH_EEPR...	17kx16x1
MC9S08DN32_FLASH_EEPR...	34kx16x1
MC9S08DN48_FLASH_EEPR...	49kx16x1
MC9S08DN60_FLASH_EEPR...	62kx16x1
MC9S08DV128_FLASH	128kx16x1
MC9S08DV16_FLASH	17kx16x1

Status
Device added.

d. You can select other devices, if required.

e. Click **Done**.

The **Add Device** dialog box closes. The devices appear in the **Flash Devices** group of the ***<target>*** **Flash Programmer Task** editor window.

10. Enter the first address from target memory used by the flash algorithm (running on the target) in the **Address** text box of the **Target RAM** group.

    a. Enter the size of the memory that the flash algorithm is allowed to use in the **Size** text box.

    b. Check the **Verify Target Memory Writes** checkbox to verify all write operations to the hardware RAM during Flash programming.

11. From the **Flash Programmer Actions** group, you can use any of the buttons listed in Table 16.1 to add various Flash Programmer actions.

Table 16.1 Task Actions and Sequence Organization

Button	Usage
Add Erase / Blank Check Action	Lets you add erase or blank check actions for flash devices
Add Program / Verify Action	Lets you add program or verify flash actions for flash devices
Add Checksum Action	Lets you add checksum actions for flash devices
Add Diagnostics Action	Lets you add a diagnostics action to the actions table.
Add Dump Flash Action	Lets you dump a portion of the flash or the entire flash.
Add Protect/Unprotect Action	Lets you modify the protection of a sector.
Remove Action	Lets you remove a sector, a group of sector, or an entire device from the **Flash Programmer Actions** table, depending on the flash capabilities.
Move Up	Lets you move a selected flash action up in the **Flash Programmer Actions** table, so that it executes before other actions beneath it in the table.
Move Down	Lets you move a selected flash action down in the **Flash Programmer Actions** table, so that the action executes after other actions above of it in the table execute.

NOTE Refer to the *Freescale Eclipse Extension Guide* for detailed documentation on the various options available in the **Flash Programmer** window.

Flash Programmer for MCF5441X

The flash programmer for MCF5441X has a specific way of writing data. The flash is comprised of multiple physical pages, each of them with the size of 0x800 bytes. At boot time the processor will read data from the first 4 pages up to 0xF80 bytes. The space between address 0xF80 and 0x2000 will not be used. Assuming we write a binary file, the data will be programmed as follows:

- Binary file address 0x0 – 0xF7F will be programmed at 0x0 – 0xF7f in flash.
- Binary file addresses higher than 0xF80 will be programmed starting from 5th physical page at 0x2000.

Because of the way how the processor boots there isn't a 1 to 1 correspondence for addresses. The pages are written with the following settings:

- Boot pages (index 0 - 3): 60 ECC bytes, in NFC_CFG register BTMD is 1 and 16BIT is 0.
- User pages (index > 3): 60 ECC bytes, in NFC_CFG register BTMD is 0 and 16BIT is 1.

Quick Access to Target Tasks

The **Target Task** toolbar helps quickly access the **Flash Programmer**, **Hardware Diagnostics**, or **Import/Export Memory** target tasks. The toolbar is populated with the last executed target tasks corresponding to the related provider and will give the possibility to faster execute any of the last executed target tasks.

Target Task Toolbar

Figure 16.10 shows the Target Task toolbar.

Figure 16.10 Target Task Toolbar

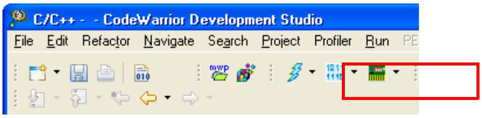

C/C++ - - CodeWarrior Development Studio
File Edit Refactor Navigate Search Project Profiler Run PE

Table 16.2 lists the icons available on the **Target Task** toolbar.

NOTE The icons provide fast access to the view, import predefined tasks, and execute recently executed tasks.

Table 16.2 Target Task Toolbar Icons

Option	Description
⚡ ▾	Enables fast access to the **Flash Programmer** target task. It also lets you access **Simple Flash** programmer to load files on target. Refer Fast Access to Flash Programmer and Tutorial L: Programming with Simple Flash.
1011 1110 ▾	Enables fast access to the **Hardware Diagnostics** target task. Refer Fast Access to Hardware Diagnostics.
▦ ▾	Enables fast access to the **Import/Export Memory** target task. Refer Fast Access to Import/Export Memory.

Fast Access to Flash Programmer

The quickly access the **Flash Programmer** target task, click the ⚡ ▾ icon.

Table 16.3 lists the available **Flash Programmer** options.

Table 16.3 Flash Programmer Options

Option	Description
Open Flash Programmer	Select to open the **Target Tasks** view.
Import Flash Task	Select to import a Flash task using the **Open** dialog box.
Flash File to Target	Select to flash file using the **Simple Flash** dialog box.

Fast Access to Hardware Diagnostics

The quickly access the **Hardware Diagnostics** target task, click the 1011 1110 ▾ icon.

Table 16.4 lists the available **Hardware Diagnostics** options.

Table 16.4 Hardware Diagnostics Options

Option	Description
Open HW Diagnostic	Select to open the **Target Tasks** view.
Import HW Diagnostic Task	Select to import a hardware diagnostics task using the **Open** dialog box.

Fast Access to Import/Export Memory

The quickly access the **Import/Export Memory** target task, click the ▦ ▾ icon.

Table 16.4 lists the available **Import/Export Memory** options.

Table 16.5 Import/Export Memory Options

Option	Description
Open Import Export Memory	Select to open the **Target Tasks** view.
Import IEM Task	Select to import a import/export memory task using the **Open** dialog box.

Flash Programmer Tutorials

This topic consists of two tutorials that demonstrate how to import and execute pre-defined target tasks. Additionally, there are tutorials on how use the ***<target>* Flash Programmer Task** editor window to create tasks, such as erasing on-chip memory, or downloading a file, and writing it into flash memory.

NOTE The Flash Programmer for RS08 works only if OSBDM connection is used.

The tutorials include:

- Tutorial A: Import and Execute HCS08 Flash Task
- Tutorial B: Import and Execute ColdFire Flash Task
- Tutorial C: Create Erase Memory Task for HCS08
- Tutorial D: Create Erase Flash Memory Task for ColdFire
- Tutorial E: Create Download Program Task for ColdFire
- Tutorial F: Import and Execute Power Architecture Flash Task
- Tutorial G: Switching Between Lock-Step and Decoupled Parallel Modes

- Tutorial H: Create and Execute Diagnostics Action Task
- Tutorial I: Dump Entire Flash
- Tutorial J: Change Protection of Sector
- Tutorial K: Fast Access to Target Tasks Editors
- Tutorial L: Programming with Simple Flash
- Tutorial M: Exporting Target Tasks

Tutorial A: Import and Execute HCS08 Flash Task

The goal of this tutorial is to select and import a pre-defined task that erases and programs the flash memory in a DEMOS908QG8, an evaluation board based on a Freescale MC9S08QG8 microcontroller.

- Import HCS08 Program Flash Task
- Execute MC9S08QG8 Task

Import HCS08 Program Flash Task

NOTE This procedure assumes that the **Target Tasks** view is visible in the perspective. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

After you have launched CodeWarrior and connected the DEMOS908QG8 board to the workstation using a USB cable, perform these steps.

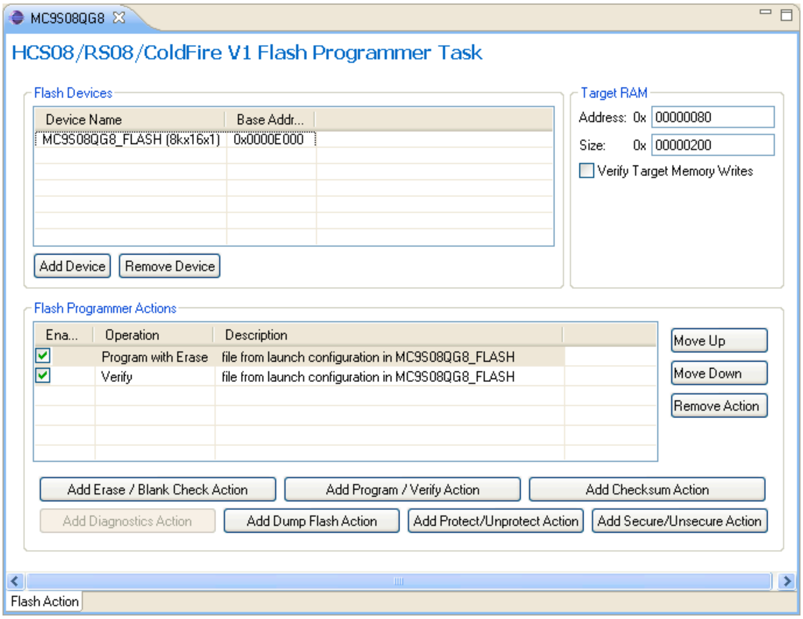
1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to `<CWInstallDir>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/HC08` and select the XML file for the board's microcontroller. For this tutorial, select `MC9S08QG8.xml`.
4. Click **Open**.

   The `MC9S08QG8` task appears in the **Target Tasks** view.
5. Double-click on the `MC9S08QG8` task to examine its contents.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the memory settings and actions for the task (Figure 16.11). Notice the actions to erase, program, and verify the contents of flash memory in the **Flash Programmer Actions** group. These actions execute in the order as they are displayed in the table, from top to bottom.

Figure 16.11 Memory Settings and Actions for MC9S08QG8 Task



MC9S08QG8
Add Device
Remove Device
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

HCS08/RS08/ColdFire V1 Flash Programmer Task

Flash Devices

Device Name	Base Addr...
MC9S08QG8_FLASH (8kx16x1)	0x0000E000

Target RAM

Address: 0x 00000080

Size: 0x 00000200

☐ Verify Target Memory Writes

Flash Programmer Actions

Ena...	Operation	Description
☑	Program with Erase	file from launch configuration in MC9S08QG8_FLASH
☑	Verify	file from launch configuration in MC9S08QG8_FLASH

Execute MC9S08QG8 Task

To execute the task, perform these steps.

1. Right-click on the `MC9S08QG8` task.
2. Select **Execute**. Alternatively, select the `MC9S08QG8` task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the DEMOS908QG8 board, erases the microcontroller’s flash memory, downloads the code, and verifies that the contents of flash match those of the file.

Congratulations! You have selected and used a target task that erased and programmed the flash memory on the MC9S08QG8 microcontroller.

Tutorial B: Import and Execute ColdFire Flash Task

The goal of this tutorial is to select and import a pre-defined task that erases and then programs the flash memory in a board based on the ColdFire MCF5213 microcontroller.

- Import MCF5213 Program Flash Task
- Import MCF5213 Program Flash Task

Import MCF5213 Program Flash Task

NOTE This tutorial assumes that you have already built a ColdFire project with the name `ColdFire_test`. It also assumes that you have previously created other tasks, so that the **Target Tasks** view is visible. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

After you have connected an MCF5213-based evaluation board to the workstation with a USB cable, perform these steps.

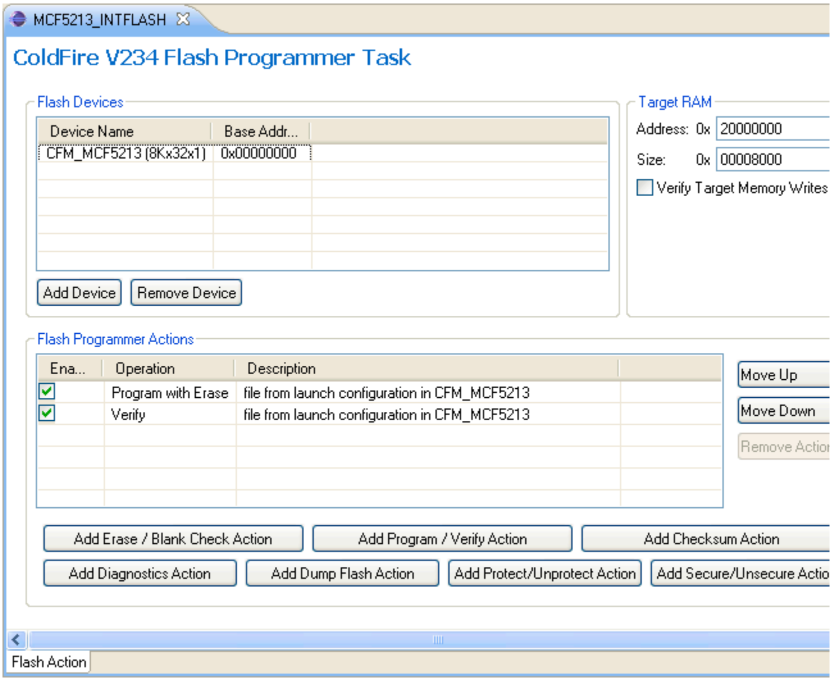
1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to `<`*`CWInstallDir`*`>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/ColdFire` and select the XML file for the board's microcontroller. For this tutorial, select `MCF5213_INTFLASH.xml`.
4. Click **Open**.

   The `MCF5213_INTFLASH` task appears in the **Target Tasks** view.
5. Double-click on the `MCF5213_INTFLASH` task to examine its contents.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the memory settings and actions for the task (Figure 16.12). Notice the actions to erase, program, and verify the contents of flash memory in the **Flash Programmer Actions** group. These actions execute in the order as they are displayed in the table, from top to bottom.

Figure 16.12 Memory Settings and Actions for the MCF5213_INTFLASH Task


MCF5213_INTFLASH
Flash Devices
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Move Up
Move Down
Remove Actio
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Actio
Flash Action

ColdFire V234 Flash Programmer Task

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

Address: 0x 20000000
Size: 0x 00008000
☐ Verify Target Memory Writes

Ena...	Operation	Description
☑	Program with Erase	file from launch configuration in CFM_MCF5213
☑	Verify	file from launch configuration in CFM_MCF5213

Execute MCF5213_INTFLASH Task

To execute the task, perform these steps.

1. Right-click on the `MCF5213_INTFLASH` task.
2. Select **Execute**. Alternatively, select the `MCF5213_INTFLASH` task and click the **Execute** icon on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the MCF5213-based board, erases the microcontroller’s flash memory, downloads the code, and verifies that the contents of flash match those of the file.

Congratulations! You have selected and used a target task that erased and programmed the flash memory on the MCF5213 microcontroller.

Tutorial C: Create Erase Memory Task for HCS08

The goal of this tutorial is to demonstrate how to create a task that erases the flash memory in a DEMOS908QG8, an evaluation board based on a Freescale MC9S08QG8 microcontroller.

- Set Up HCS08 Erase Task
- Execute HCS08 Erase Task

Set Up HCS08 Erase Task

NOTE This procedure assumes that you have already created a project named `DEMO9S08QG8_test`. It also assumes that you have previously created other tasks, so that the **Target Tasks** view is visible. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

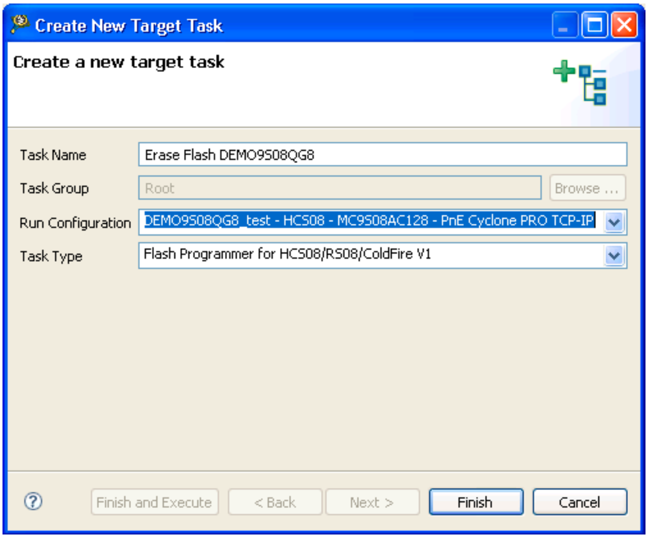
After you have launched CodeWarrior and connected the DEMOS908QG8 board to the host system using a USB cable, perform these steps.

1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **New Task**.

   The **Create New Target Task** wizard appears.
3. In the **Task Name** text box, enter the name of the target task. For example, enter `Erase Flash DEMO9S08QG8`.
4. From the **Run Configuration** drop-down list, select a run configuration. For example, select `DEMO9S08QG8_test - HCS08 - MC9S08AC128 - PnE Multilink_Cyclone PRO TCP-IP`.
5. From the **Task Type** drop-down list, select `Flash Programmer for HCS08/ColdFire V1` from the listbox.

   The dialog box should appear as in Figure 16.13.

Figure 16.13 Task Settings for Erasing Flash on MC9S08QG8



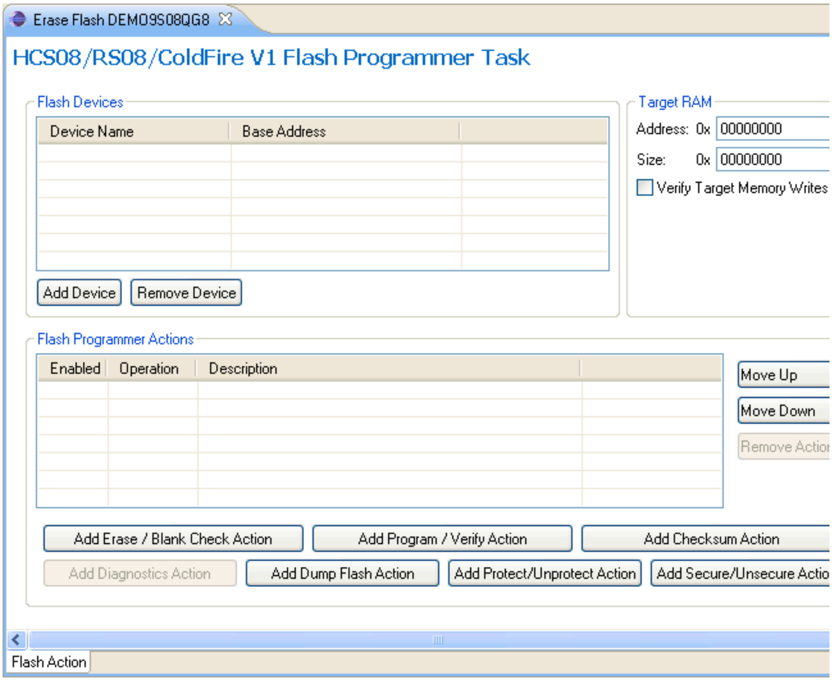
Create New Target Task
Create a new target task
Finish and Execute
< Back
Next >
Finish
Cancel
Browse ...

Task Name: Erase Flash DEMO9S08QG8
Task Group: Root
Run Configuration: DEMO9S08QG8_test - HCS08 - MC9S08AC128 - PnE Cyclone PRO TCP-IP
Task Type: Flash Programmer for HCS08/RS08/ColdFire V1

6. Click **Finish** to create the task.

   The **<*target*> Flash Programmer Task** editor window appears, as shown in Figure 16.14.

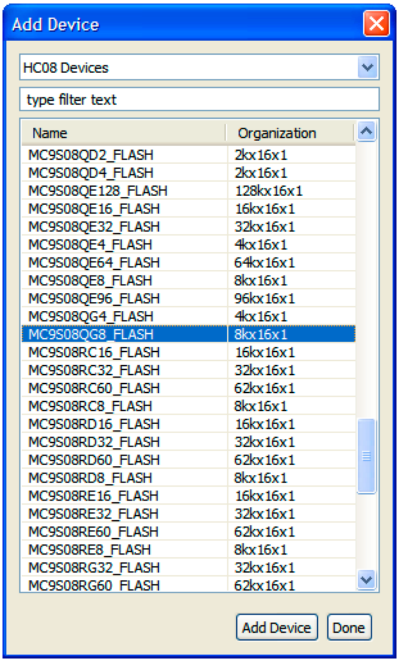
Figure 16.14 ***<target>*** **Flash Programmer Task Editor Window to Erase HCS08 Flash**


Erase Flash DEMO9S08QG8
HCS08/RS08/ColdFire V1 Flash Programmer Task
Flash Devices
Device Name
Base Address
Add Device
Remove Device
Target RAM
Address: 0x 00000000
Size: 0x 00000000
☐ Verify Target Memory Writes
Flash Programmer Actions
Enabled
Operation
Description
Move Up
Move Down
Remove Actio
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Actio
Flash Action

7. Click **Add Device**.

   The **Add Device** dialog box appears (Figure 16.15).

Figure 16.15 Add Device for HCS08 Derivative



HC08 Devices

type filter text

Name	Organization
MC9S08QD2_FLASH	2kx16x1
MC9S08QD4_FLASH	2kx16x1
MC9S08QE128_FLASH	128kx16x1
MC9S08QE16_FLASH	16kx16x1
MC9S08QE32_FLASH	32kx16x1
MC9S08QE4_FLASH	4kx16x1
MC9S08QE64_FLASH	64kx16x1
MC9S08QE8_FLASH	8kx16x1
MC9S08QE96_FLASH	96kx16x1
MC9S08QG4_FLASH	4kx16x1
MC9S08QG8_FLASH	8kx16x1
MC9S08RC16_FLASH	16kx16x1
MC9S08RC32_FLASH	32kx16x1
MC9S08RC60_FLASH	62kx16x1
MC9S08RC8_FLASH	8kx16x1
MC9S08RD16_FLASH	16kx16x1
MC9S08RD32_FLASH	32kx16x1
MC9S08RD60_FLASH	62kx16x1
MC9S08RD8_FLASH	8kx16x1
MC9S08RE16_FLASH	16kx16x1
MC9S08RE32_FLASH	32kx16x1
MC9S08RE60_FLASH	62kx16x1
MC9S08RE8_FLASH	8kx16x1
MC9S08RG32_FLASH	32kx16x1
MC9S08RG60_FLASH	62kx16x1

8. Scroll through the list of microcontroller derivatives and select `MC9S08QG8_FLASH`. You can also type the filter text in the given text box.
9. Click **Add Device**.
10. Click **Done**.

    You return to the ***<target>*** **Flash Programmer Task** editor window, with the selected device appearing in the **Flash Devices** group. The **Target RAM** group displays the start address of RAM memory, and its size.

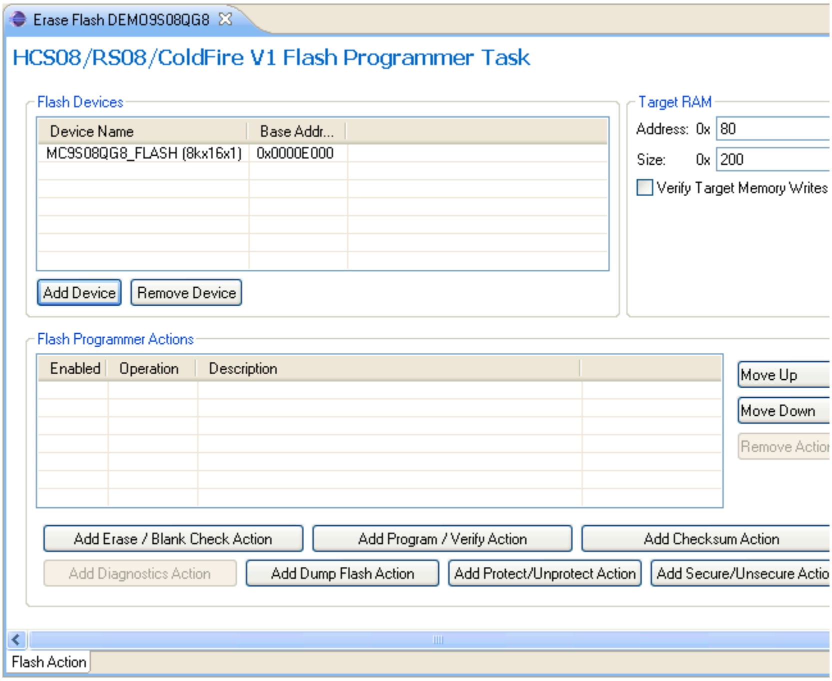
11. Click **Add Erase / Blank Check Action**.

    The **Add Erase / Blank Check Action** dialog box appears. It displays the flash devices added in the task and their base addresses.

12. Click **Add Erase Action**.
13. Click **Done**.

    You return to the ***<target>*** **Flash Programmer Task** editor window, and the action appears in the **Flash Programmer Actions** group (Figure 16.15).

Figure 16.16 *<target>* Flash Programmer Task Editor Window Settings for DEMO9S08QG8



Erase Flash DEMO9S08QG8

HCS08/RS08/ColdFire V1 Flash Programmer Task

Flash Devices

Device Name	Base Addr...
MC9S08QG8_FLASH (8kx16x1)	0x0000E000

Target RAM

Address: 0x 80

Size: 0x 200

☐ Verify Target Memory Writes

Flash Programmer Actions

Enabled	Operation	Description

Add Device
Remove Device
Move Up
Move Down
Remove Actio
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Actio
Flash Action

14. Click the ☒ icon to save the settings and close the ***<target>* Flash Programmer Task** editor window.

    You can access the newly-made `Erase Flash DEMO9S08QG8` task from the **Target Tasks** view.

Execute HCS08 Erase Task

To erase the Flash memory on the MC9S08QG8, you use the Erase Flash DEMO9S08QG8 task that you made in the previous topic.

To execute the task and erase the memory, perform these steps.

1. Go the **Target Tasks** view and right-click on the `Erase Flash DEMO9S08QG8` task.
2. Select **Execute**. Alternatively, select the `Erase Flash DEMO9S08QG8` task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.
3. In the **Console** view, status messages appear as the IDE connects to the board and erases the memory.

Congratulations! You have erased the on-chip Flash memory in the DEMO9S08QG8 board’s microcontroller.

Tutorial D: Create Erase Flash Memory Task for ColdFire

The goal of this tutorial is to demonstrate how to use the ***<target>* Flash Programmer Task** editor window to create a task that erases specific topics of Flash memory in a ColdFire MCF5213.

- Set Up ColdFire Erase Task
- Execute ColdFire Erase Task

Set Up ColdFire Erase Task

NOTE This tutorial assumes that you have already built a ColdFire project with the name `ColdFire_test`. It also assumes that you have previously created other tasks, so that the **Target Tasks** view is visible. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

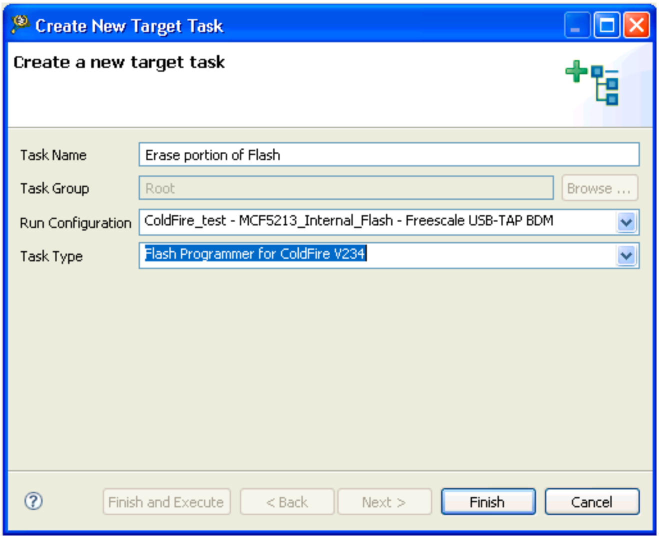
After you have connected an MCF5213-based evaluation board to the workstation with a USB cable, perform these steps.

1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **New Task**.

   The **Create New Target Task** wizard appears.
3. In the **Task Name** text box, enter the name of the target task. For example, enter `Erase portion of Flash`.
4. From the **Run Configuration** drop-down list, select a run configuration. For example, select `ColdFire_test - MCF52213 Internal Flash - Freescale USB-TAP BDM`.
5. From the **Task Type** drop-down list, select `Flash Programmer for ColdFire V234` from the listbox.

   The **Create New Target Task** wizard should appear as in Figure 16.17.

Figure 16.17 Task Settings for Erasing Flash on MCF5213



Create New Target Task
Create a new target task

Task Name: Erase portion of Flash

Task Group: Root

Run Configuration: ColdFire_test - MCF5213_Internal_Flash - Freescale USB-TAP BDM

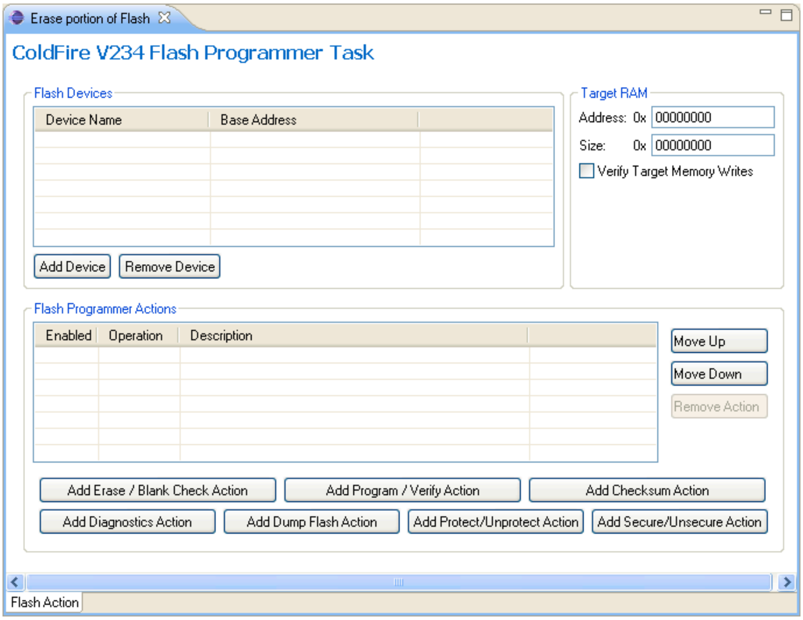
Task Type: Flash Programmer for ColdFire V234

Browse ...
Finish and Execute
< Back
Next >
Finish
Cancel

6. Click **Finish** to create the task.

The ***<target>*** **Flash Programmer Task** editor window appears, as shown in Figure 16.18.

Figure 16.18 *<target>* Flash Programmer Task Editor Window

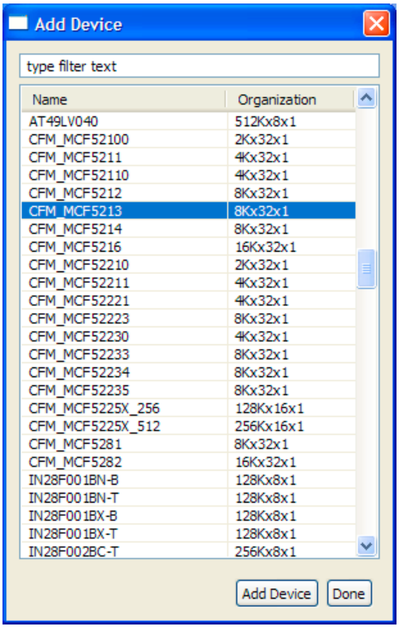

Erase portion of Flash
ColdFire V234 Flash Programmer Task
Flash Devices
Device Name
Base Address
Add Device
Remove Device
Target RAM
Address: 0x
Size: 0x
Verify Target Memory Writes
Flash Programmer Actions
Enabled
Operation
Description
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

Address: 0x 00000000
Size: 0x 00000000
☐ Verify Target Memory Writes

7. Click **Add Device**.

   The **Add Device** dialog box appears (Figure 16.19).

Figure 16.19 Add Device for ColdFire Derivative



Name	Organization
AT49LV040	512Kx8x1
CFM_MCF52100	2Kx32x1
CFM_MCF5211	4Kx32x1
CFM_MCF52110	4Kx32x1
CFM_MCF5212	8Kx32x1
CFM_MCF5213	8Kx32x1
CFM_MCF5214	8Kx32x1
CFM_MCF5216	16Kx32x1
CFM_MCF52210	2Kx32x1
CFM_MCF52211	4Kx32x1
CFM_MCF52221	4Kx32x1
CFM_MCF52223	8Kx32x1
CFM_MCF52230	4Kx32x1
CFM_MCF52233	8Kx32x1
CFM_MCF52234	8Kx32x1
CFM_MCF52235	8Kx32x1
CFM_MCF5225X_256	128Kx16x1
CFM_MCF5225X_512	256Kx16x1
CFM_MCF5281	8Kx32x1
CFM_MCF5282	16Kx32x1
IN28F001BN-B	128Kx8x1
IN28F001BN-T	128Kx8x1
IN28F001BX-B	128Kx8x1
IN28F001BX-T	128Kx8x1
IN28F002BC-T	256Kx8x1

8. Scroll through the list of microcontroller derivatives and select `CFM_MCF5213`.
9. Click **Add Device**.
10. Click **Done**.

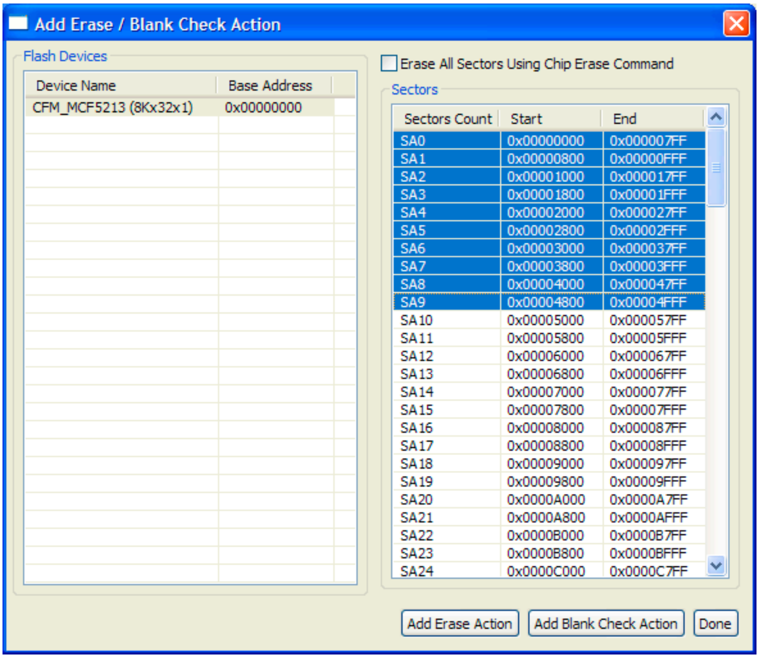
    You return to the ***<target>*** **Flash Programmer Task** editor window, with the selected device appearing in the **Flash Devices** group.

11. Click **Add Erase / Blank Check Action**.

    The **Add Erase / Blank Check Action** dialog box appears. It displays the flash devices and their addresses.

12. In the **Sectors** group, click on the desired start address of the flash memory, then shift-click on the end address to select all of the sectors of Flash memory you want erased (Figure 16.20).

Figure 16.20 Add Erase / Blank Action Dialog Box for ColdFire



Device Name	Base Address
CFM_MCF5213 (8Kx32x1)	0x00000000

☐ Erase All Sectors Using Chip Erase Command

Sectors Count	Start	End
SA0	0x00000000	0x000007FF
SA1	0x00000800	0x00000FFF
SA2	0x00001000	0x000017FF
SA3	0x00001800	0x00001FFF
SA4	0x00002000	0x000027FF
SA5	0x00002800	0x00002FFF
SA6	0x00003000	0x000037FF
SA7	0x00003800	0x00003FFF
SA8	0x00004000	0x000047FF
SA9	0x00004800	0x00004FFF
SA10	0x00005000	0x000057FF
SA11	0x00005800	0x00005FFF
SA12	0x00006000	0x000067FF
SA13	0x00006800	0x00006FFF
SA14	0x00007000	0x000077FF
SA15	0x00007800	0x00007FFF
SA16	0x00008000	0x000087FF
SA17	0x00008800	0x00008FFF
SA18	0x00009000	0x000097FF
SA19	0x00009800	0x00009FFF
SA20	0x0000A000	0x0000A7FF
SA21	0x0000A800	0x0000AFFF
SA22	0x0000B000	0x0000B7FF
SA23	0x0000B800	0x0000BFFF
SA24	0x0000C000	0x0000C7FF

13. Click **Add Erase Action**.
14. Click **Done**.

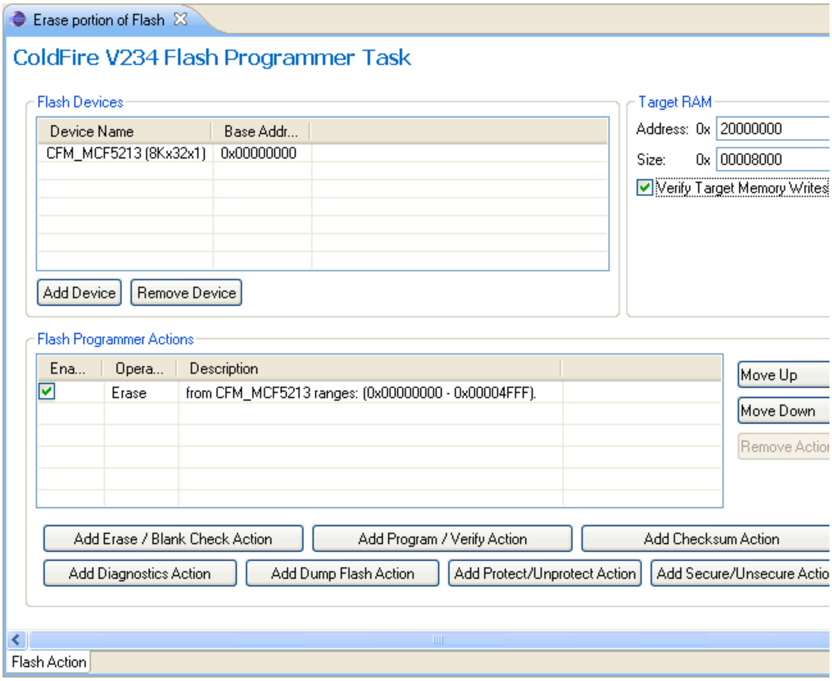
NOTE To erase of all Flash memory at the same time, check **Erase All Sectors Using Chip Erase Command**.

You return to the ***<target>* Flash Programmer Target** editor window, and the action appears in the **Flash Programmer Actions** group.

15. To allocate a buffer of RAM to hold the erasure algorithm.
    a. Enter the start address of the RAM buffer in **Address** text box of the **Target RAM** group.
    b. In the **Size** option, enter the amount of RAM that makes up the buffer.
    c. Check the **Verify Target Memory Writes** checkbox. For this example, the start address of the buffer was `0X20000000`, and its size was `0X00008000`.

       The **Target RAM** group displays the start address of the RAM memory buffer, and its size (Figure 16.21).

Figure 16.21 Flash Programmer Task Editor Window with ColdFire Erase Settings


Erase portion of Flash
ColdFire V234 Flash Programmer Task
Flash Devices
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Move Up
Move Down
Remove Actio
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Actio
Flash Action

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

Address: 0x 20000000

Size: 0x 00008000

☑ Verify Target Memory Writes

Ena...	Opera...	Description
☑	Erase	from CFM_MCF5213 ranges: (0x00000000 - 0x00004FFF).

16. Click the ☒ icon to save the settings and close the ***<target>* Flash Programmer Task** editor window.

    You can access the newly-made `Erase portion of Flash` task from the **Target Tasks** view.

Execute ColdFire Erase Task

To erase the Flash memory on the MCF5213, you use the task that you made in the previous topic.

To execute the task, perform these steps.

1. Go the **Target Tasks** view.
2. Right-click on the `Erase portion of Flash` task.

3. Select **Execute**. Alternatively, click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   In the **Console** view, status messages appear as the IDE connects to the board and erases the flash memory.

You have erased the selected on-chip Flash memory sectors in the MCF5123 microcontroller.

Tutorial E: Create Download Program Task for ColdFire

The goal of this tutorial is to demonstrate how to create a task that downloads a program into the ColdFire microcontroller's flash memory before being debugged.

- Set Up Download Task
- Execute ColdFire Program Task

Set Up Download Task

NOTE This tutorial assumes that you have already built a ColdFire project with the name `ColdFire_test`. It also assumes that you have created the `Erase portion of Flash` task from the previous tutorial.

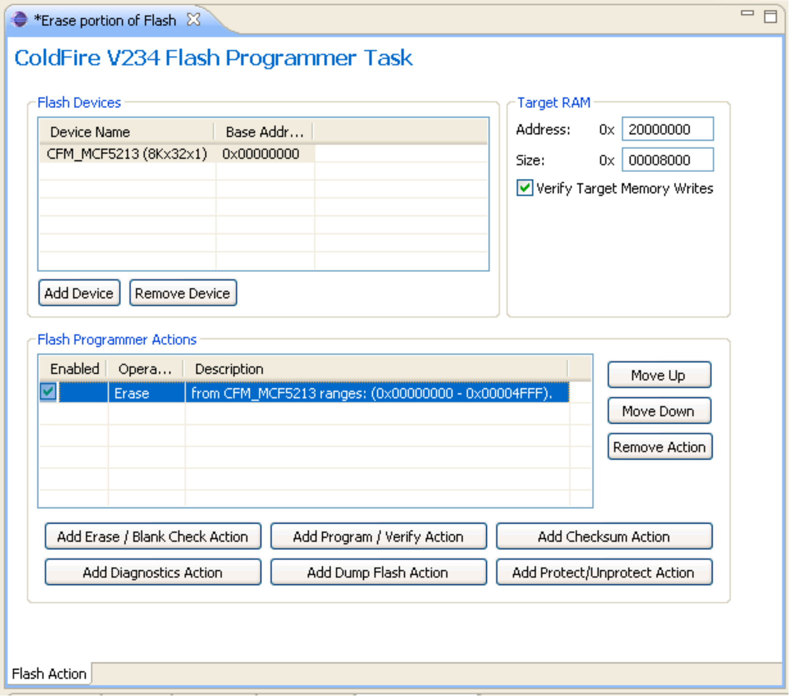
WARNING! To avoid the microcontroller getting caught in an indeterminant state, it is important that its flash memory be erased before attempting to program it. The erase memory action of this task will erase the microcontroller's flash memory before the second action, created in this topic, downloads a program into it. Do not attempt to program flash memory without erasing it first.

After you have connected an MCF5213-based evaluation board to the workstation with a USB cable, perform these steps.

1. Go to the **Target Tasks** view in either the **C/C++ or Debug** perspective.
2. Right-click on this view and double-click on the task `Erase portion of Flash` that you made in the previous topic.

   The **ColdFire V234 Flash Programmer Task** editor window appears (Figure 16.22).

Figure 16.22 Task Settings for Erasing Memory on MCF5213


*Erase portion of Flash
ColdFire V234 Flash Programmer Task
Flash Devices
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Flash Action

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

Address: 0x 20000000

Size: 0x 00008000

☑ Verify Target Memory Writes

Enabled	Opera...	Description
☑	Erase	from CFM_MCF5213 ranges: (0x00000000 - 0x00004FFF).

3. Allocate a buffer of RAM that holds the programming algorithm, along with any program code as it is written into flash.
   a. Enter the start address of the RAM buffer in **Address** text box of the **Target RAM** group.
   b. In the **Size** text box, enter the amount of RAM that makes up the buffer.
   c. Check the **Verify Target Memory Writes** option. For this example, the start address of the buffer was `0X20000000`, and its size was `0X00008000`.

NOTE Since this configuration was taken care of when the Erase action was set up, you do not have to enter anything for this step. However, it is described here for the sake of completeness.

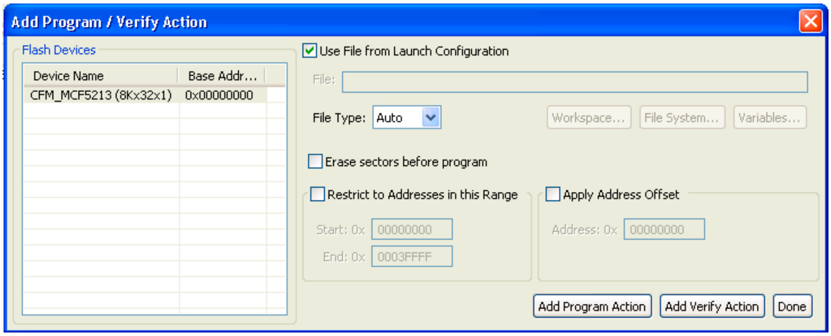
The **Target RAM** group displays the start address of the RAM memory buffer, and its size.

4. Click **Add Program / Verify Action**.

   The **Add Program / Verify Action** dialog box appears, with the ColdFire device selected.

5. Check **Use File from Launch Configuration** to use the default `.elf` file made by the project (Figure 16.23).

Figure 16.23 Adding File to Add Program / Verify Action Dialog Box



Add Program / Verify Action

Flash Devices

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

☑ Use File from Launch Configuration

File:

File Type: Auto

☐ Erase sectors before program

☐ Restrict to Addresses in this Range

Start: 0x 00000000

End: 0x 0003FFFF

☐ Apply Address Offset

Address: 0x 00000000

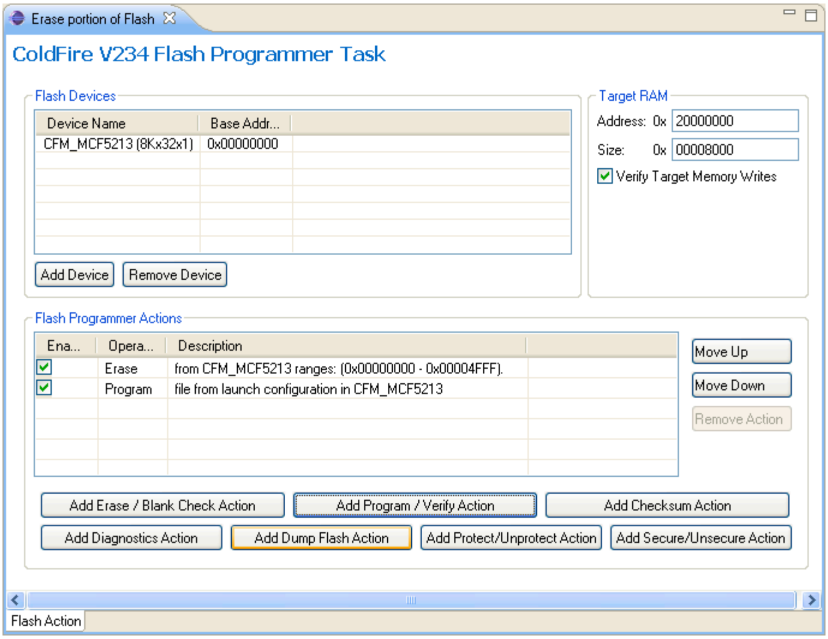
6. Click **Add Program Action**.

7. Click **Done**.

   You return to the ***<target>*** **Flash Programmer Target** editor window, with the program action appearing in the **Flash Programmer Actions** group (Figure 16.24).

NOTE If you want to download a file other than the launch configuration's default file, click on the **Workspace**, **File System**, or **Variables** button and navigate to the file.

Figure 16.24 Settings for Downloading Program to ColdFire Microcontroller



Erase portion of Flash
Flash Devices
Target RAM
Add Device
Remove Device
Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

ColdFire V234 Flash Programmer Task

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

Address: 0x 20000000
Size: 0x 00008000
☑ Verify Target Memory Writes

Ena...	Opera...	Description
☑	Erase	from CFM_MCF5213 ranges: (0x00000000 - 0x00004FFF).
☑	Program	file from launch configuration in CFM_MCF5213

8. Click the ☒ icon to save the settings and close the ***<target>* Flash Programmer Task** editor window.

   The revised `Erase portion of Flash` task is available from the **Target Tasks** view.

Execute ColdFire Program Task

To execute this task, perform these steps.

1. Go the **Target Tasks** view and right-click on the `MCF5213_INTFLASH` task.
2. Select **Execute**. Alternatively, select the `MCF5213_INTFLASH` task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.
3. In the **Console** view, status messages appear as the IDE connects to the board and erases the memory.

Congratulations! You have erased the on-chip Flash memory in the DEMO9S08QG8 board’s microcontroller.

Tutorial F: Import and Execute Power Architecture Flash Task

The goal of this tutorial is to select and import a pre-defined Power Architecture flash task.

- Import Power Architecture Program Flash Task
- Execute Predefined Task

Import Power Architecture Program Flash Task

To import a pre-defined Power Architecture flash task, perform these steps.

1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to `<CWInstallDir>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/E200` and select the XML file for the board's microcontroller.

NOTE One processor has multiple flash areas that are treated as different devices. Therefore, you should be very sure of the needs when selecting a task.

There are predefined target tasks for each flash module. However, depending on processor type some of them might not be available.

- Internal Flash — The Internal Flash target task is for Code Flash and Data Flash if they exist or for the user area of Internal Flash. Comprises everything - the area for application (sometimes has code + data), shadow flash and tester areas.
  - Code Flash — Program's code should be programmed here.
  - Data flash — Flash special design to hold data.
- Shadow Flash — Flash that contains system configuration options besides the user area. Erasing it can also change the system state after reset. Some of these registers are for device's security and you cannot write anything there. Additionally, the erase sequence is different. After you erase the device, the security information is programmed in the flash and thus the blank check fails.
- Tester Areas — Special flash that cannot be erased and also have registers that can change processor's behavior like which flash sectors are protected. Changing them cannot be reverted, which means that the Flash is written once.

NOTE Some processors can interpret two sets of instructions: BOOKE and VLE. If the processor supports both of them target tasks are provided for each instruction set. The user should choose the right one.

4. Click **Open**.

   The selected task appears in the **Target Tasks** view.

5. Double-click on the task to examine its contents.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the memory settings and actions for the task. Notice the actions to erase, program, and/or verify the contents of flash memory in the **Flash Programmer Actions** group.

Execute Predefined Task

To execute the task, perform these steps.

1. Right-click on the imported task.
2. Select **Execute**. Alternatively, select the imported task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the board, erases the microcontroller's flash memory, downloads the code, and verifies that the contents of flash match those of the file.

Congratulations! You have selected and used a target task that erased and programmed the flash memory on the pre-defined Power Architecture task.

Tutorial G: Switching Between Lock-Step and Decoupled Parallel Modes

The goal of this tutorial is to demonstrate the steps to switch between the **Lock-Step Mode (LSM)** and **Decoupled Parallel Mode (DPM)**.

- Import DPM Target Task
- Execute Predefined DPM Task
- Hardware Reset

Import DPM Target Task

NOTE This procedure assumes that the **Target Tasks** view is visible in the perspective. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

After you have launched CodeWarrior and the machine has a USB NEXUS Multilink connected to a Leopard (for example, MPC5643L) DPM board, perform these steps.

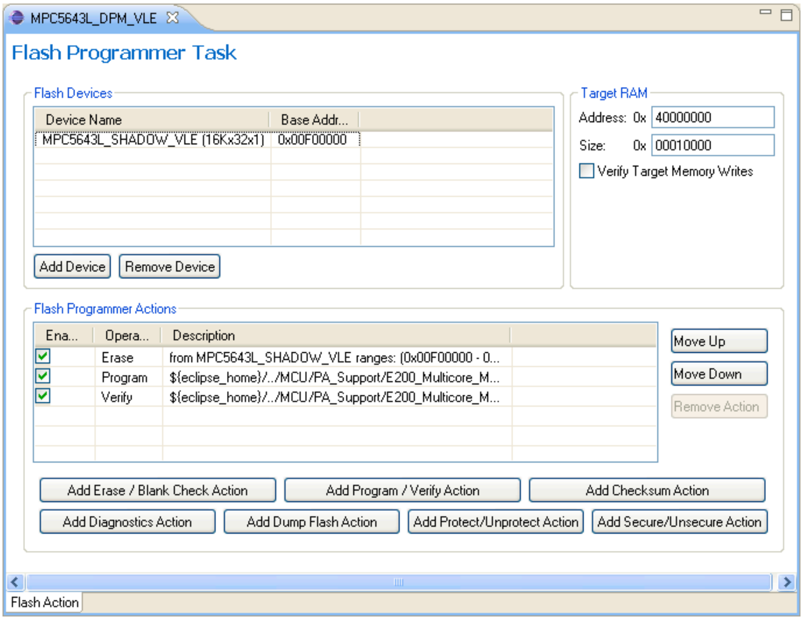
1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to `<CWInstallDir>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/E200` and select the XML file for the board's microcontroller. For this tutorial, select `MPC5643L_DPM_VLE.xml`.
4. Click **Open**.

   The `MPC5643L_DPM_VLE` task appears in the **Target Tasks** view.
5. Double-click on the `MPC5643L_DPM_VLE` to examine its contents.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the memory settings and actions for the task (Figure 16.25). Notice the actions to erase, program, and verify the contents of flash memory in the **Flash Programmer Actions** group. These actions execute in the order as they are displayed in the table, from top to bottom.

Figure 16.25 Settings for MPC5643L_DPM_VLE Target Task


MPC5643L_DPM_VLE
Flash Programmer Task
Flash Devices
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

Device Name	Base Addr...
MPC5643L_SHADOW_VLE (16Kx32x1)	0x00F00000

Address: 0x 40000000
Size: 0x 00010000
☐ Verify Target Memory Writes

Ena...	Opera...	Description
☑	Erase	from MPC5643L_SHADOW_VLE ranges: (0x00F00000 - 0...
☑	Program	${eclipse_home}/../MCU/PA_Support/E200_Multicore_M...
☑	Verify	${eclipse_home}/../MCU/PA_Support/E200_Multicore_M...

Execute Predefined DPM Task

To execute the task, perform these steps.

1. Right-click on the imported task.
2. Select **Execute**. Alternatively, select the imported task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the board, erases the microcontroller's flash memory, downloads the code, and verifies that the contents of flash match those of the file.

Hardware Reset

To reset the hardware, perform these steps.

1. Unplug the power cord.
2. Plug the power cord again.

   or

   Simply press the reset button on the hardware or target board.

Congratulations! You have selected and used a target task that erased and programmed the flash memory on the pre-defined Power Architecture task.

Import LSM Target Task

NOTE This procedure assumes that the **Target Tasks** view is visible in the perspective. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

After you have launched CodeWarrior and the machine has a USB NEXUS Multilink connected to a Leopard (for example, MPC5643L) LSM board, perform these steps.

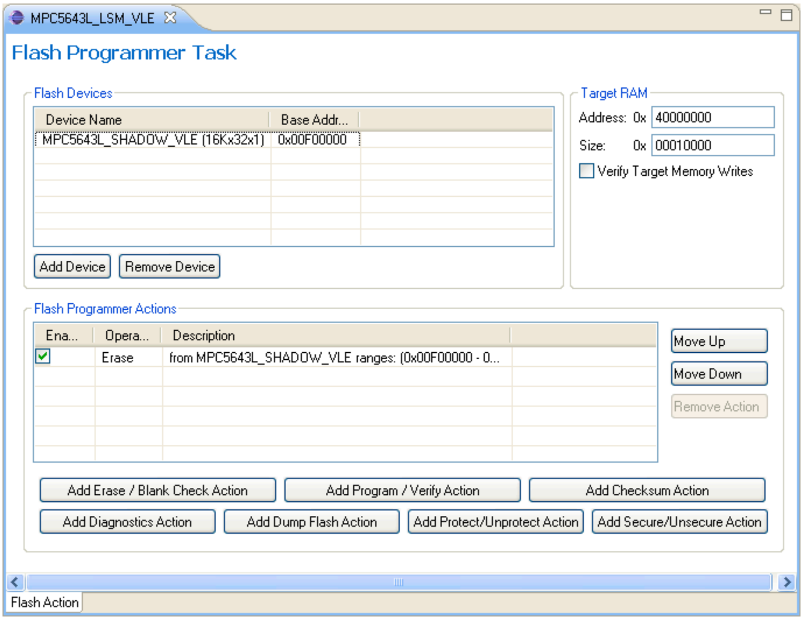
1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click on this view and select **Import**. Alternatively, click the 📥 icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to `<`*`CWInstallDir`*`>/MCU/bin/plugins/support/TargetTask/Flash_Programmer/E200` and select the XML file for the board's microcontroller. For this tutorial, select `MPC5643L_LSM_VLE.xml.`
4. Click **Open**.

   The `MPC5643L_LSM_VLE` task appears in the **Target Tasks** view.
5. Double-click on the `MPC5643L_LSM_VLE` to examine its contents.

   The ***<target>*** **Flash Programmer Task** editor window appears, and displays the memory settings and actions for the task (Figure 16.26). Notice the actions to erase the contents of flash memory in the **Flash Programmer Actions** group. These actions execute in the order as they are displayed in the table, from top to bottom.

Figure 16.26 Settings for MPC5643L_LSM_VLE Target Task



MPC5643L_LSM_VLE
Flash Programmer Task
Flash Devices
Add Device
Remove Device
Target RAM
Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

Device Name	Base Addr...
MPC5643L_SHADOW_VLE (16Kx32x1)	0x00F00000

Address: 0x 40000000

Size: 0x 00010000

☐ Verify Target Memory Writes

Ena...	Opera...	Description
☑	Erase	from MPC5643L_SHADOW_VLE ranges: (0x00F00000 - 0...

Execute Predefined LSM Task

To execute the task, perform these steps.

1. Right-click on the imported task.
2. Select **Execute**. Alternatively, select the imported task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the board, erases the microcontroller's flash memory.

Congratulations! You have switched between the Lock-Step and Decoupled Parallel Modes.

Tutorial H: Create and Execute Diagnostics Action Task

The goal of this tutorial is to demonstrate the steps to create and execute the diagnostics action using the flash programmer.

- Set Up Diagnostics Action Task

- Execute Diagnostics Action Task

Set Up Diagnostics Action Task

NOTE This procedure assumes that you have imported a pre-defined Flash Programmer task in the **Target Tasks** view. It also assumes that you have previously created other tasks, so that the **Target Tasks** view is visible. If it is not visible, perform the steps in the Working with Flash Programmer topic to open the **Target Tasks** view.

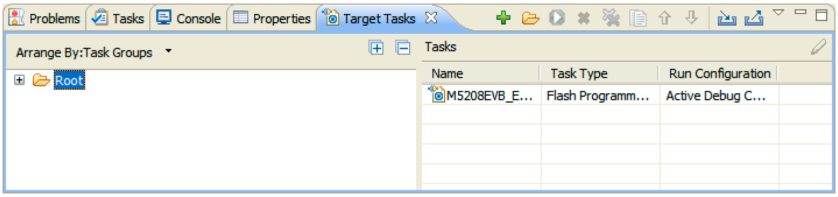
After you have launched CodeWarrior and connected the `M52277EVB_SPI` board to the host system using a USB cable, perform these steps.

1. Go to the **Target Tasks** view in either the **C/C++** or **Debug** perspective.
2. Right-click in the **Target Tasks** view and select **Import**. Alternatively, click the icon on the **Target Tasks** view toolbar.

   The **Open** dialog box appears.
3. Navigate to the pre-defined tasks folder at `<CW MCU install>\MCU\bin\plugins\support\TargetTask\Flash_Programmer\` and select the desired `.xml` file for your hardware target. For example, select `M52277EVB_SPI.xml.`
4. Click **Open**.

   The selected task appears in the **Target Tasks** view (Figure 16.27).

Figure 16.27 Pre-defined Task in Target Tasks View



Name	Task Type	Run Configuration
M5208EVB_E...	Flash Programm...	Active Debug C...

NOTE The predefined erase/program tasks are not mandatory for diagnostics.

5. Right-click on the task’s name and select **Execute**.

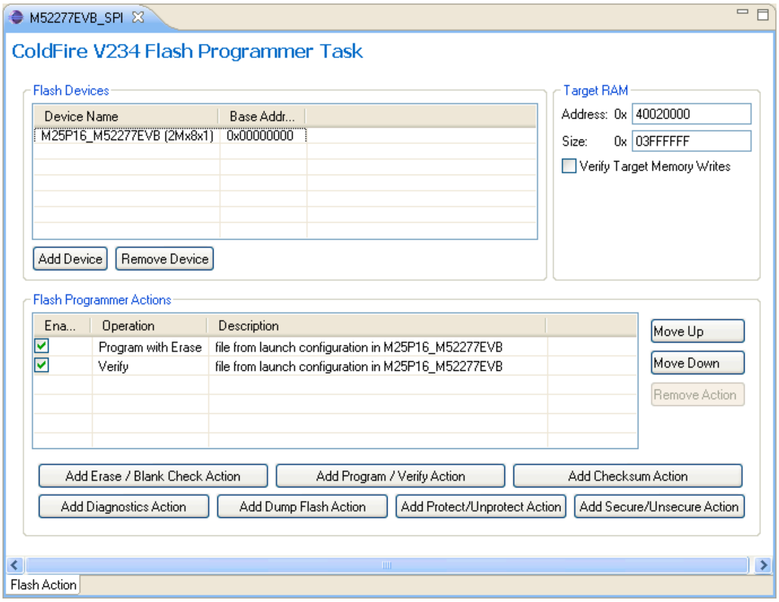
   The task’s Flash Programmer actions execute in sequence. First, they erase the hardware target’s flash memory. Next, they check whether the flash is correctly erased. If true, they program the file’s code into the flash and check if it programmed without errors.

NOTE When a predefined flash programmer task is imported, its **Run Configuration** is set as Active Debug Context. If the task is imported and there is no active debug session, then the ▶ icon will be disabled as in Figure 16.27. Associate the selected target task to a different Run Configuration to enable the ▶ icon.

6. Double-click on the task name, to examine the task stored Flash Programmer actions.

   The ***<target>*** **Flash Programmer Task** editor window appears, as shown in Figure 16.28.

Figure 16.28 *<target>* Flash Programmer Task Editor Window


M52277EVB_SPI
ColdFire V234 Flash Programmer Task
Flash Devices
Target RAM
Add Device
Remove Device
Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action
Flash Action

Device Name	Base Addr...
M25P16_M52277EVB (2Mx8x1)	0x00000000

Address: 0x 40020000
Size: 0x 03FFFFFF
☐ Verify Target Memory Writes

Ena...	Operation	Description
☑	Program with Erase	file from launch configuration in M25P16_M52277EVB
☑	Verify	file from launch configuration in M25P16_M52277EVB

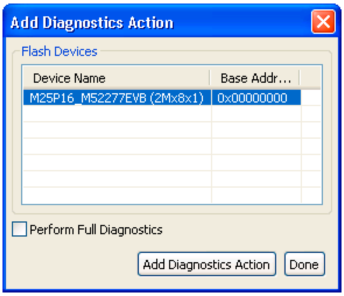
7. Click **Add Diagnostics Action**.

   The **Add Diagnostics Action** dialog box appears (Figure 16.29). It displays the flash devices and the base addresses.

NOTE The full diagnostics does the same thing as diagnostics but also prints the blank status for sectors. It can take significantly longer to complete.

NOTE If more than one flash is available in **Flash Devices** table, the **Add Diagnostics Action** table lets you select the flash where you want to run the diagnostics.

Figure 16.29 Add Diagnostics Action Dialog Box


Add Diagnostics Action
Flash Devices

Device Name	Base Addr...
M25P16_M52277EVB (2Mx8x1)	0x00000000

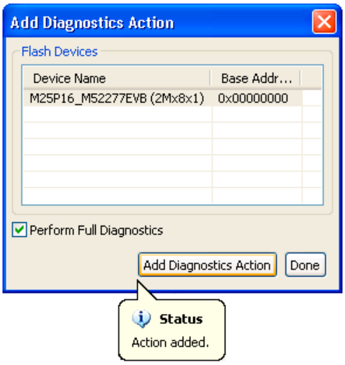
☐ Perform Full Diagnostics

Add Diagnostics Action
Done

8. Check the **Perform Full Diagnostics** checkbox if you want to perform complete diagnostics on the selected flash device.
9. Click the **Add Diagnostics Actions** button (Figure 16.29).

   You get a popup with a status that the device is added (Figure 16.30).

Figure 16.30 Add Diagnostics Actions Dialog Box — Popup with Status


Add Diagnostics Action
Flash Devices

Device Name	Base Addr...
M25P16_M52277EVB (2Mx8x1)	0x00000000

☑ Perform Full Diagnostics

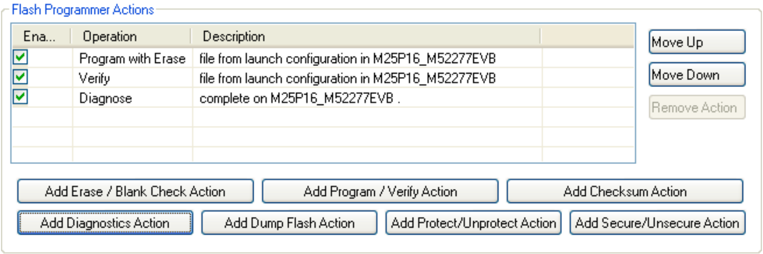
Add Diagnostics Action
Done

Status
Action added.

10. Click **Done**.

    The **Add Diagnostics Action** dialog box closes. The action appears in the **Flash Programmer Actions** table of the ***<target>*** **Flash Programmer Task** editor window (Figure 16.31).

Figure 16.31 Flash Programmer Actions Table



Ena...	Operation	Description
☑	Program with Erase	file from launch configuration in M25P16_M52277EVB
☑	Verify	file from launch configuration in M25P16_M52277EVB
☑	Diagnose	complete on M25P16_M52277EVB .

Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action

Execute Diagnostics Action Task

To execute the task, perform these steps.

1. Right-click on the `M52277EVB_SPI` task.
2. Select **Execute**. Alternatively, select the `M52277EVB_SPI` task and click the **Execute** icon ▶ on the **Target Tasks** view toolbar.

   CodeWarrior establishes contact with the `M52277EVB_SPI` board, erases the microcontroller's flash memory, downloads the code, verifies, and diagnoses the contents on `M52277EVB_SPI`.

You have created and executed a diagnostic action task on the `M52277EVB_SPI` microcontroller.

Tutorial I: Dump Entire Flash

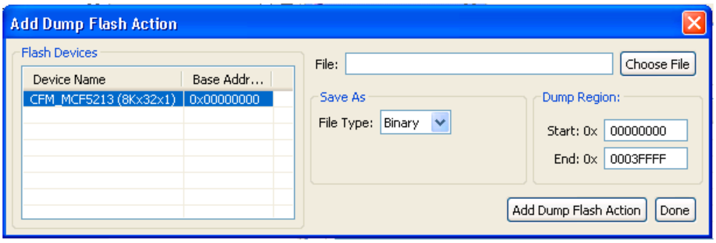
The goal of this tutorial is to demonstrate how to dump selected sectors of a flash device or the entire flash device.

To add a dump flash action:

1. Click the **Dump Flash Action** button.

   The **Add Dump Flash Action** dialog box (Figure 16.32) appears.

Figure 16.32 Add Dump Flash Action Dialog Box


Add Dump Flash Action
Flash Devices
Choose File
Save As
Dump Region:
Add Dump Flash Action
Done

Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

File:

File Type: Binary

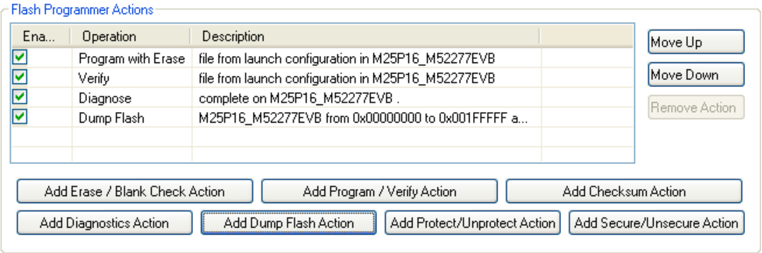
Start: 0x 00000000

End: 0x 0003FFFF

2. Specify the file name in the **File** text box. The flash is dumped in this selected file.
3. Select the file type from the **File Type** drop-down list. You can select any one of the following file types:
    - Srec — Saves files in Motorola S-record format.
    - Binary — Saves files in binary file format.
4. Specify the memory range for which you want to add dump flash action.
    - Specify the start address of the range in the **Start** text box.
    - Specify the end address of the range in the **End** text box.
5. Click **Add Dump Flash Action**.
6. Click **Done**.

   The **Add Dump Flash Action** dialog box closes and the added dump flash action appear in the **Flash Programmer Actions** table in the **Flash Programmer Task** editor window (Figure 16.33).

Figure 16.33 Added Dump Flash Actions


Flash Programmer Actions
Move Up
Move Down
Remove Action
Add Erase / Blank Check Action
Add Program / Verify Action
Add Checksum Action
Add Diagnostics Action
Add Dump Flash Action
Add Protect/Unprotect Action
Add Secure/Unsecure Action

Ena...	Operation	Description
☑	Program with Erase	file from launch configuration in M25P16_M52277EVB
☑	Verify	file from launch configuration in M25P16_M52277EVB
☑	Diagnose	complete on M25P16_M52277EVB .
☑	Dump Flash	M25P16_M52277EVB from 0x00000000 to 0x001FFFFF a...

Tutorial J: Change Protection of Sector

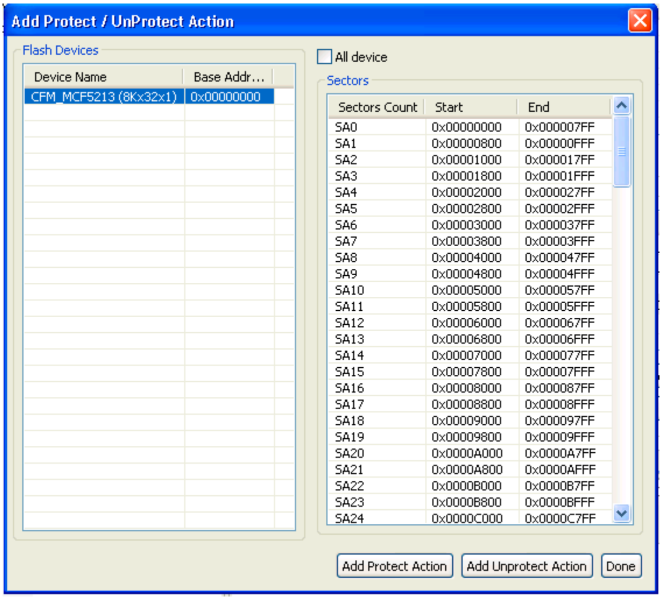
The goal of this tutorial is to demonstrate how to protect / unprotect actions enable you to change the protection of a sector in the flash device.

To add a protect / unprotect action:

1. Click the **Add Protect/Unprotect Action** button.

   The **Add Protect/Unprotect Action** dialog box (Figure 16.34) appears.

Figure 16.34 Add Protect / Unprotect Action Dialog Box



Device Name	Base Addr...
CFM_MCF5213 (8Kx32x1)	0x00000000

Sectors Count	Start	End
SA0	0x00000000	0x000007FF
SA1	0x00000800	0x00000FFF
SA2	0x00001000	0x000017FF
SA3	0x00001800	0x00001FFF
SA4	0x00002000	0x000027FF
SA5	0x00002800	0x00002FFF
SA6	0x00003000	0x000037FF
SA7	0x00003800	0x00003FFF
SA8	0x00004000	0x000047FF
SA9	0x00004800	0x00004FFF
SA10	0x00005000	0x000057FF
SA11	0x00005800	0x00005FFF
SA12	0x00006000	0x000067FF
SA13	0x00006800	0x00006FFF
SA14	0x00007000	0x000077FF
SA15	0x00007800	0x00007FFF
SA16	0x00008000	0x000087FF
SA17	0x00008800	0x00008FFF
SA18	0x00009000	0x000097FF
SA19	0x00009800	0x00009FFF
SA20	0x0000A000	0x0000A7FF
SA21	0x0000A800	0x0000AFFF
SA22	0x0000B000	0x0000B7FF
SA23	0x0000B800	0x0000BFFF
SA24	0x0000C000	0x0000C7FF

2. Select a sector from the **Sectors** table and click the **Add Protect Action** button to add a protect operation on the selected sector.

NOTE Press **CTRL** or **SHIFT** keys for selecting multiple sectors from the **Sectors** table.

---

3. Click the **Add Unprotect Action** button to add an unprotect action on the selected sector.

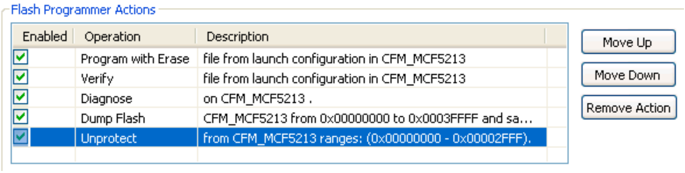
NOTE Check the **All Device** checkbox to add action on full device.

---

4. Click **Done**.

   The **Add Protect/ Unprotect Action** dialog box closes and the added protect or unprotect actions appear in the **Flash Programmer Actions** table in the **Flash Programmer Task** editor window (Figure 16.35).

Figure 16.35 Protect / Unprotect Action



Flash Programmer Actions

Enabled	Operation	Description
☑	Program with Erase	file from launch configuration in CFM_MCF5213
☑	Verify	file from launch configuration in CFM_MCF5213
☑	Diagnose	on CFM_MCF5213 .
☑	Dump Flash	CFM_MCF5213 from 0x00000000 to 0x0003FFFF and sa...
☑	Unprotect	from CFM_MCF5213 ranges: (0x00000000 - 0x00002FFF).

Tutorial K: Fast Access to Target Tasks Editors

The goal of this tutorial is to demonstrate managing and fast accessing of the target tasks' editors used by the Target Task framework.

You can access target tasks by:

- Editing tasks in a project. Refer Editing Tasks in Project.
- Editing tasks imported in a previous session. Refer Editing Tasks Imported in Previous Session.
- Storing task to a file. Refer Storing Task to File.

Editing Tasks in Project

NOTE Before editing ensure that the target task has been added to the project by a wizard or another method and has an extension recognized by the feature (.ttf).

To edit tasks in a project:

1. Double-click on the file in the project.
2. The appropriate task editor appears.
3. The task is imported in the Target Task Framework.

Editing Tasks Imported in Previous Session

NOTE Before editing ensure that a target task has been imported and the previous Eclipse session has been closed.

To edit tasks imported in a previous session:

1. Open the **Target Tasks** view.
2. Double-click to open the desired task.

   The target task editor appears.
3. Make the appropriate changes and close the target task editor.

NOTE Changing the task will also save changes to the file in the project. The save is done to the file in project only if the task has been imported from a project (with double click). Otherwise, it will prompt to save the file.

Storing Task to File

NOTE Before storing the task ensure that the target task has been imported or created and appropriate changes have been made.

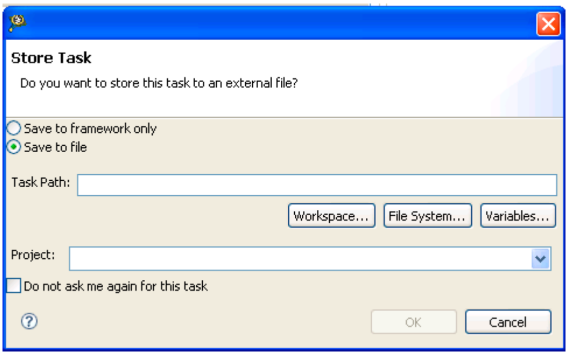
To store task to a file:

1. Open the **Target Tasks** view.
2. Double-click to open the desired task.

   The target task editor appears.
3. Make the appropriate changes and save the target task.

   The **Store Task** dialog box appears (Figure 16.36).

Figure 16.36 Store Task Dialog Box



Store Task

Do you want to store this task to an external file?

○ Save to framework only
● Save to file

Task Path:

Project:

☐ Do not ask me again for this task

4. Select the **Save to File** option.

5. In the **Task Path** text box, specify the path where you want to store the task. You can use the **Workspace**, **File System**, or **Variables** buttons to navigate to the desired location.
6. From the **Project** drop-down list select the project where you want to store you target task.

NOTE Check the **Do not ask me again for this task** checkbox to save these settings for the current target task.

7. Click **OK**.

   The dialog box closes and associates the file to the specified project and saves in target task framework and not necessarily in the project.

NOTE The above mentioned feature has a preference that will not display the save as dialog and always save in target task framework. The settings are located in **Windows > Preferences > C/C++ >Debug > CodeWarrior Debugger > Show "Save As"** dialog box when saving a new task.

Tutorial L: Programming with Simple Flash

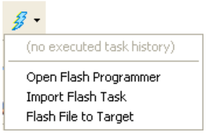
The goal of this tutorial is to demonstrate the use of CodeWarrior Simple Flash Programmer. This feature enables you to perform these basic flash operations:

- Erasing Flash Device
- Programming File

To open the **Simple Flash** dialog box:

1. Click the **Flash Programmer** icon on the IDE toolbar (Figure 16.38).

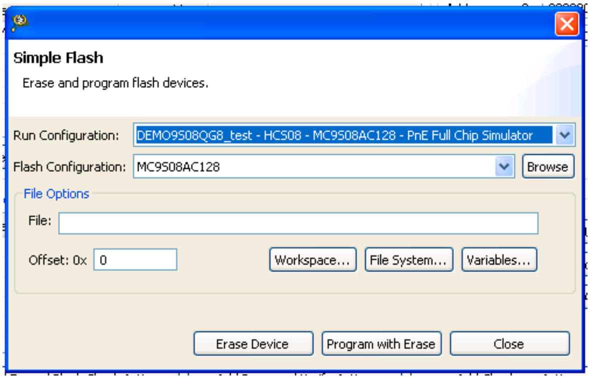
Figure 16.37 Flash Programmer Icon



2. Select **Flash File to Target**.

   The **Simple Flash** dialog box appears (Figure 16.38).

Figure 16.38 Simple Flash Dialog Box



Simple Flash

Erase and program flash devices.

Run Configuration: DEMO9S08QG8_test - HCS08 - MC9S08AC128 - PnE Full Chip Simulator

Flash Configuration: MC9S08AC128

File:

Offset: 0x 0

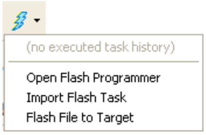
- **Run Configuration** drop-down list — Lists all run configurations defined in Eclipse. If a connection to the target has already been made the control becomes inactive and contains the text Active Debug Configuration.
- **Flash Configuration** drop-down list — Lists predefined target tasks for the processor selected in the Launch Configuration and tasks added by user with the **Browse** button. The values in this drop-down list are updated based on the processor selected in the launch configuration. For more information on launch configurations, see *<product> targeting manual*.
- **File Options** group — Allows selecting the file to be programmed on the flash device and the location.
  - **File** text box — Enables you to specify the filename. You can use the **Workspace**, **File System**, or **Variables** buttons to select the desired file.
  - **Offset** text box — Enables you to specify offset location for a file. If no offset is specified the default value of zero is used. The offset is always added to the start address of the file. If the file doesn't contain address information then zero is considered as start address.
- **Erase Device** button — Erases the flash device. In case you have multiple flash blocks on the device, all blocks are erased. If you want to selectively erase or program blocks, use the Flash Programmer feature.
- **Program with Erase** button — Erases the sectors that are occupied with data and then programs the file. If the flash device can not be accessed at sector level then the flash device is completely erased.

Erasing Flash Device

To erase a flash device, perform these steps.

1. Click the **Flash Programmer** icon on the IDE toolbar (Figure 16.39).

Figure 16.39 Flash Programmer Icon


(no executed task history)
Open Flash Programmer
Import Flash Task
Flash File to Target

2. Select **Flash File to Target**.
3. The **Simple Flash** dialog box appears (Figure 16.39).
4. Select a run configuration from the **Run Configuration** drop-down list.

NOTE If a connection is already established with the target, this control is disabled.

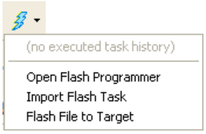
The Flash Configuration drop-down list is updated with the supported configurations for the processor from the launch configuration.

5. Select a flash configuration from the **Flash Configuration** drop-down list.
6. Click the **Erase Device** button.

Programming File

1. Click the **Flash Programmer** icon on the IDE toolbar (Figure 16.40).

Figure 16.40 Flash Programmer Icon


(no executed task history)
Open Flash Programmer
Import Flash Task
Flash File to Target

2. Select **Flash File to Target**.
3. The **Simple Flash** dialog box appears (Figure 16.38).
4. Select a run configuration from the **Run Configuration** drop-down list.

NOTE If a connection is already established with the target, this control is disabled.

The Flash Configuration drop-down list is updated with the supported configurations for the processor from the launch configuration.

5. Select a flash configuration from the **Flash Configuration** drop-down list.

6. Specify the file name in the **File** text box. You can use the **Workspace**, **File System**, or **Variables** buttons to select the desired file.
7. Specify the offset location in the **Offset** text box.
8. Click the **Program with Erase** button.

Tutorial M: Exporting Target Tasks

The goal of this tutorial is to demonstrate how to export a target task to an external file.

To export a target task:

1. Select the target task in the **Target Task** view.
2. Click the **Export** button from the **Target Task** view toolbar. Alternatively, right-click the target task and select **Export** from the context menu.

   The **Save As** dialog box appears.
3. Specify a file name in the **File name** drop-down list.
4. Click **Save**.

   The exported task is stored in XML format.

Working with Hardware Diagnostics Window

The **Hardware Diagnostics** window lets you run a series of diagnostic tests that determine if the basic hardware is functional.

These tests include:

- Memory read/write — Makes a read / write access to memory in order to read or write a byte, word (2 bytes), and long word (4 bytes) to / from memory.
- Scope loop — Makes read and write accesses to memory in a loop at target address. The the loop speed settings determine the time between accesses. The loop can only be stopped by cancelling the test.
- Memory tests — Requires you to set the access size and target address from the access settings group and the settings present in the Memory Tests group.

On the Eclipse IDE, the hardware diagnostics feature runs like a target task.

To create a hardware diagnostic target task:

1. Click **Window** > **Show View** > **Other**.

   The **Show View** dialog box appears.
2. Expand the **Debug** group and select **Target Tasks**.

3. Click **OK**.
4. Click + from the **Target Tasks** view **t**oolbar to create a new target task.

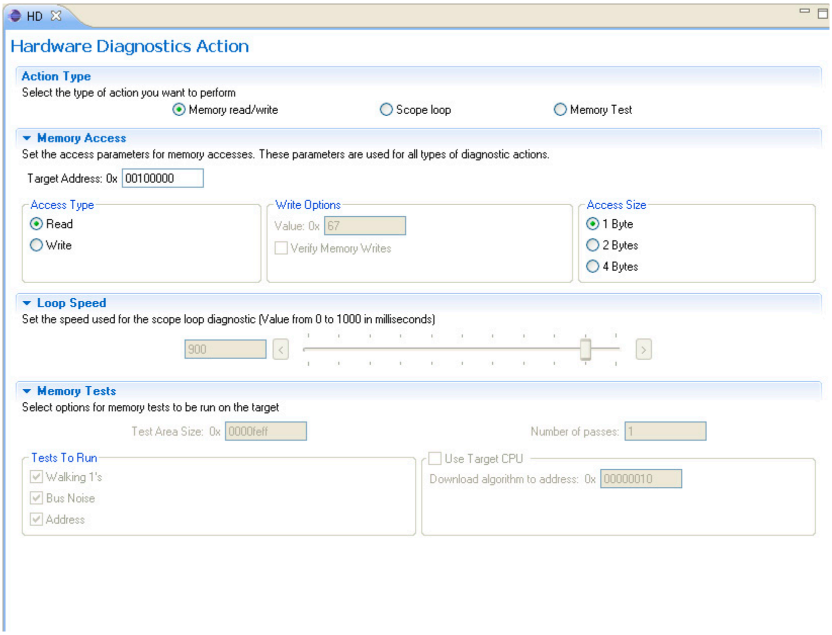
   The **Create New Target Task** wizard appears.
5. In the **Task Name** text box, enter the name of the target task.
6. From the **Run Configuration** drop-down list, select a configuration.

NOTE Select Active Debug Context from the **Run Configuration** drop-down list, if you want to use hardware diagnostics over an active debugger session, else select any of the specified debug context from the list.

7. From the **Task Type** drop-down list, select **Hardware Diagnostic**.
8. Click **Finish**.

   The **Hardware Diagnostics Action** window appears as seen in Figure 16.41.

Figure 16.41 Hardware Diagnostics Action Window



Hardware Diagnostics Action

Action Type

Select the type of action you want to perform

● Memory read/write ○ Scope loop ○ Memory Test

Memory Access

Set the access parameters for memory accesses. These parameters are used for all types of diagnostic actions.

Target Address: 0x 00100000

Access Type: ● Read ○ Write

Write Options: Value: 0x 67 ☐ Verify Memory Writes

Access Size: ● 1 Byte ○ 2 Bytes ○ 4 Bytes

Loop Speed

Set the speed used for the scope loop diagnostic (Value from 0 to 1000 in milliseconds)

900

Memory Tests

Select options for memory tests to be run on the target

Test Area Size: 0x 0000feff

Number of passes: 1

Tests To Run: ☑ Walking 1's ☑ Bus Noise ☑ Address

☐ Use Target CPU

Download algorithm to address: 0x 00000010

The **Hardware Diagnostics Action** window includes the following groups:

- **Action Type** — Used to set various action types. The options you select in this group enables the options in the other groups of the window.

- **Memory Access** — Configures diagnostic tests for performing memory reads and writes over the remote connection interface.
- **Loop Speed** — Configures diagnostic tests for performing repeated memory reads and writes over the remote connection interface.
- **Memory Tests** — Configures the memory tests that you can run on the target.

NOTE The **Use Target CPU** group appears grayed-out and is not applicable for the HCS08 and RS08 Target Tasks.

NOTE Refer the *Freescale Eclipse Extensions Guide* for detailed documentation of the various options available in the **Hardware Diagnostics Action** window.

Manipulating Target Memory

You can manipulate the target's memory in these ways:

- Import — Read encoded data from a specified file, decode that data into a specific format, and copy the decoded data into a specified memory range. For information, refer Creating Target Task to Import Memory.
- Export — Read data from a specified memory range, encode that data in a specific format, and store the encoded data in an output file. For information, refer Creating Target Task to Export Memory.
- Fill — Fill a specified memory range with a specific data pattern. For information, refer Fill Memory with Data Pattern.

Creating Target Task to Import Memory

Perform these steps to create a target task to import memory:

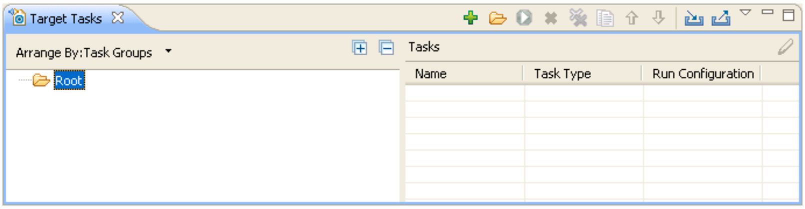
1. Select **Window** > **Show View** > **Other**.

   The **Show View** dialog box appears.

2. From the **Debug** group, select **Target Tasks**.

   The **Target Tasks** view appears (Figure 16.42)

Figure 16.42 Target Tasks View


Target Tasks
Arrange By:Task Groups
Root
Tasks
Name
Task Type
Run Configuration

3. Right-click in the **Target Tasks** view and select **New Task** from the context menu.

   The **Create New Target Task** wizard appears (Figure 16.43).

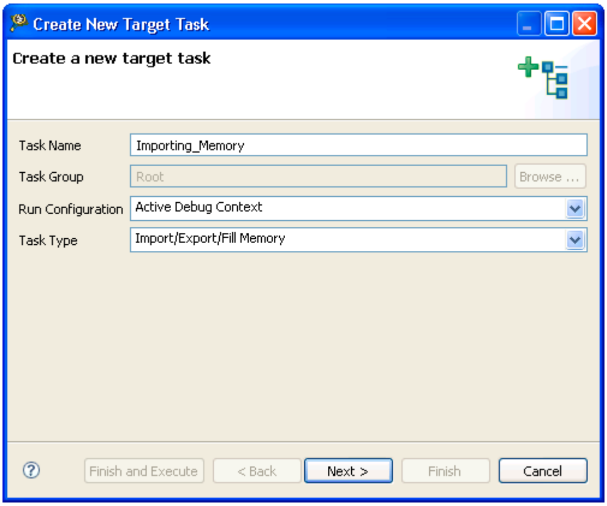
4. In the **Task Name** text box, enter a name for the new task. For example, *Importing_Memory*.

5. Use the **Run Configuration** list box to specify the configuration that the task launches and uses to connect the target. For example, select *Active Debug Context*.

NOTE If the task does not successfully launch the configuration that you specify, the **Execute** button of the **Target Tasks** view toolbar stays disabled.

6. From the **Task Type** list box, select *Import/Export/Fill Memory*.

Figure 16.43 Create New Target Task Wizard


Create New Target Task
Create a new target task

Task Name: Importing_Memory
Task Group: Root
Run Configuration: Active Debug Context
Task Type: Import/Export/Fill Memory

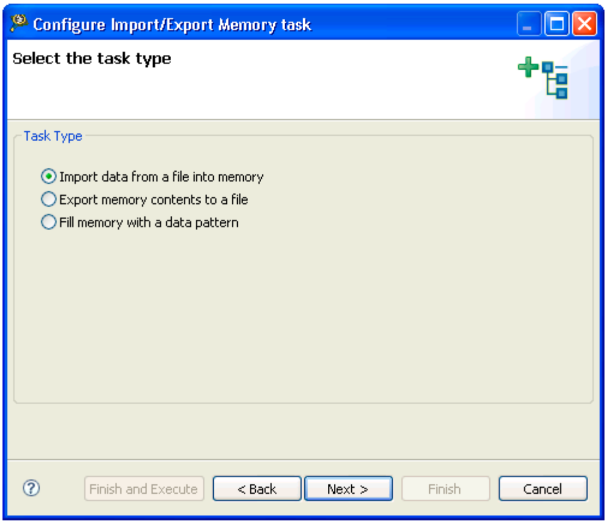
Browse ...
Finish and Execute
< Back
Next >
Finish
Cancel

7. Click **Next**.

   The **Select the task type** page (Figure 16.44) of the **Configure Import/Export Memory task** dialog box appears.

8. Select **Import data from a file into memory** task type option.

Figure 16.44 Create New Target Task Wizard — Select Task Type



Configure Import/Export Memory task

Select the task type

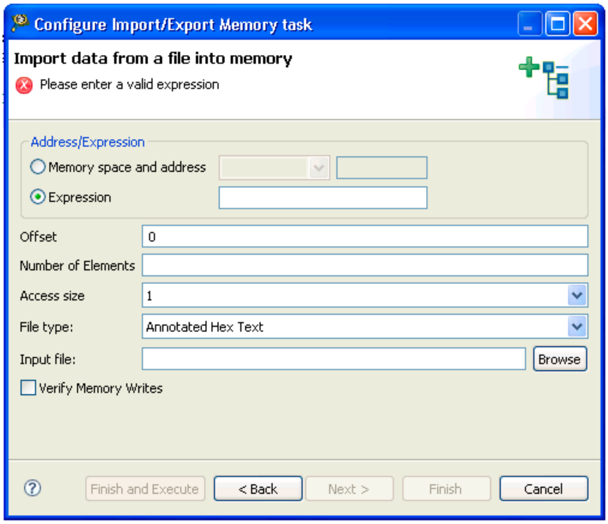
Task Type

● Import data from a file into memory
○ Export memory contents to a file
○ Fill memory with a data pattern

9. Click **Next**.

   The **Import data from a file into memory** page (Figure 16.45) of the **Configure Import/Export Memory task** dialog box appears. This page lets you read encoded data from a user specified file, decode it, and copy it into a user specified memory range.

Figure 16.45 Import data from a file into memory Page



Configure Import/Export Memory task

Import data from a file into memory

☒ Please enter a valid expression

Address/Expression

○ Memory space and address

● Expression

Offset: 0

Number of Elements:

Access size: 1

File type: Annotated Hex Text

Input file:

☐ Verify Memory Writes

10. Specify options as explained in Table 16.6.

NOTE CodeWarrior IDE validates information as you enter it. If there are errors, a message appears near the page title.

Table 16.6 Import Data from a File into memory Page Options

Item	Description
Memory space and address	Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.
Expression	Enter the memory address or expression at which the data transfer starts.
Offset	Enter an value to offset addresses contained in Motorola S-Record and Annotated Hex Text data formats. The field remains disabled for all other data formats.
Number of Elements	Enter the total number of elements to be transferred.

Table 16.6 Import Data from a File into memory Page Options (*continued*)

Item	Description
Access size	Denotes the number of addressable units of memory that the debugger accesses in transferring one data element. The default values shown are 1, 2, 4, and 8 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.
File type	Defines the format in which the wizard encodes the data it imports. By default, the following file types are supported: • Annotated Hex Text • Hex Text • Motorola S-Record • Raw Binary • Signed Decimal Text • Unsigned decimal Text
Input File	Enter the path to the file that contains the data to be imported. Click the **Browse** button to select the import file through the standard **File Open** dialog box.
Verify Memory Writes	Check the option to verify success of each data write to the memory.

11. Click **Finish**.

    CodeWarrior IDE saves your changes, closes the **Configure Import/Export Memory task** wizard, and displays the newly created import task in the **Tasks** list of the **Target Tasks** view.

NOTE Alternatively, click **Finish and Execute** to save your changes and execute the newly created import task immediately.

Creating Target Task to Export Memory

Perform these steps to create a target task to export memory:

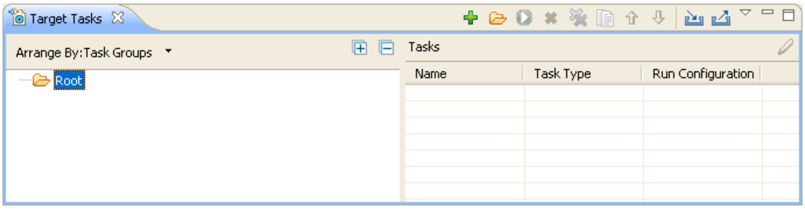
1. Select **Window** > **Show View** > **Other**.

   The **Show View** dialog box appears.

2. From the **Debug** group, select **Target Tasks**.

   The **Target Tasks** view appears (Figure 16.46)

Figure 16.46 Target Tasks View



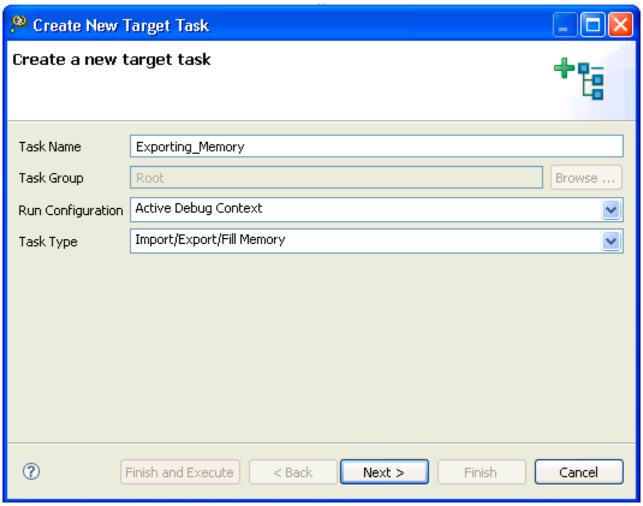
3. Right-click in the **Target Tasks** view and select **New Task** from the context menu.

   The **Create New Target Task** wizard appears (Figure 16.47).
4. In the **Task Name** text box, enter a name for the new task. For example, *Exporting_Memory*.
5. Use the **Run Configuration** list box to specify the configuration that the task launches and uses to connect the target. For example, *Active Debug Context*.

NOTE If the task does not successfully launch the configuration that you specify, the **Execute** button of the **Target Tasks** view toolbar stays disabled.

6. From the **Task Type** list box, select *Import/Export/Fill Memory*.

Figure 16.47 Create New Target Task Wizard



Create New Target Task

Create a new target task

Task Name: Exporting_Memory

Task Group: Root

Run Configuration: Active Debug Context

Task Type: Import/Export/Fill Memory

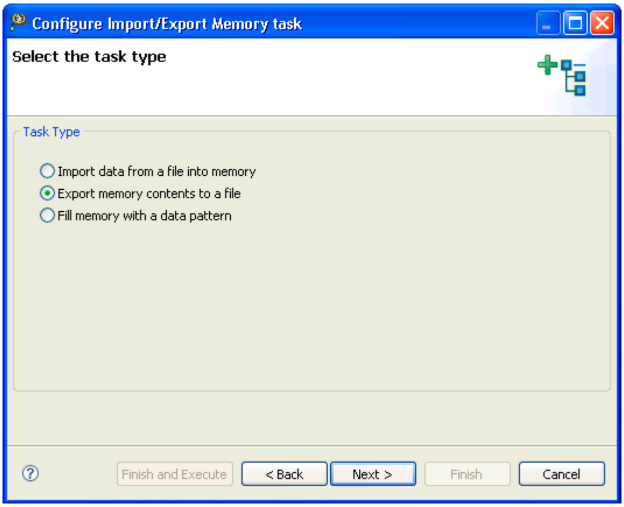
Browse ...
Finish and Execute
< Back
Next >
Finish
Cancel

7. Click **Next**.

   The **Select the task type** page (Figure 16.48) appears.

8. Select **Export data from memory into a file** task type option.

Figure 16.48 Create New Target Task Wizard — Select Task Type



Configure Import/Export Memory task

Select the task type

Task Type

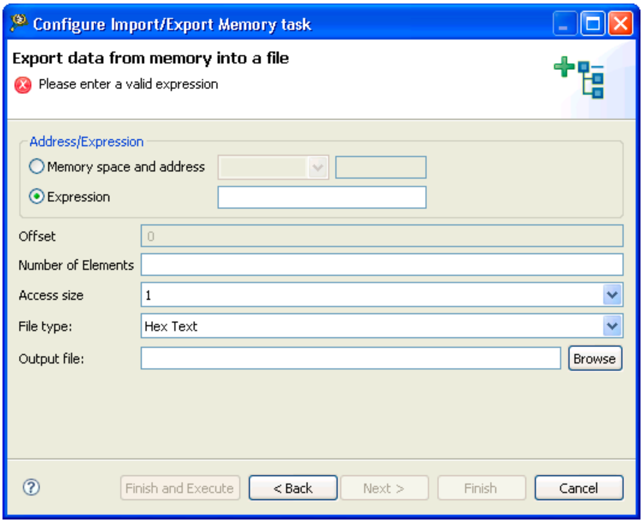
○ Import data from a file into memory
● Export memory contents to a file
○ Fill memory with a data pattern

Finish and Execute
< Back
Next >
Finish
Cancel

9. Click **Next**.

   The **Export data from memory into a file** page (Figure 16.49) appears. This page lets you read data from a user specified memory range, encode it in a user specified format, and store this encoded data in a user specified output file.

Figure 16.49 Export data from memory into a file Page


Configure Import/Export Memory task

Export data from memory into a file

Please enter a valid expression

Address/Expression
○ Memory space and address
● Expression

Offset: 0
Number of Elements:
Access size: 1
File type: Hex Text
Output file:

10. Specify options as explained in Table 16.7.

NOTE CodeWarrior IDE validates information as you enter it. If there are errors, a message appears near the page title.

Table 16.7 Export Data from Memory into a File Options

Item	Description
Memory space and address	Enter the literal address and memory space on which the data transfer is performed. The Literal address field allows only decimal and hexadecimal values.
Expression	Enter the memory address or expression at which the data transfer starts.

Table 16.7 Export Data from Memory into a File Options

Item	Description
Offset	Enter a value to offset addresses contained in Motorola S-Record and Annotated Hex Text data formats. The field remains disabled for all other data formats.
Number of Elements	Enter the total number of elements to be transferred.
Access Size	Denotes the number of addressable units of memory that the debugger accesses in transferring one data element. The default values shown are 1, 2, 4, and 8 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.
File Type	Defines the format in which the wizard encodes the data it imports. By default, the following file types are supported: • Annotated Hex Text • Hex Text • Motorola S-Record • Raw Binary • Signed Decimal Text • Unsigned decimal Text
Output File	Enter the path of the file to which the wizard will write data. Click the **Browse** button to save the export file through the standard **File Save As** dialog box.

11. Click **Finish**.

    CodeWarrior IDE saves your changes, closes the **Configure Import/Export Memory task** wizard, and displays the newly created export task in the **Tasks** list of the **Target Tasks** view.

NOTE Alternatively, click **Finish and Execute** to save your changes and execute the newly created export task immediately.

Fill Memory with Data Pattern

To fill memory with a specified data pattern, perform these steps.

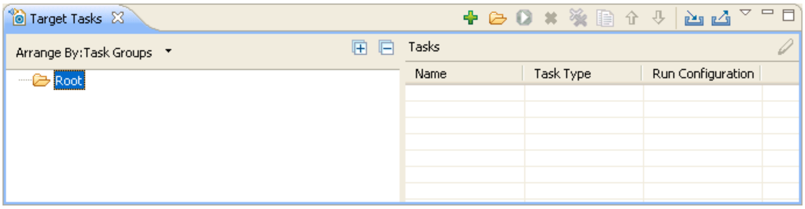
1. Select **Window** > **Show View** > **Other**.

   The **Show View** dialog box appears.

2. From the **Debug** group, select **Target Tasks**.

   The **Target Tasks** view appears (Figure 16.50)

Figure 16.50 Target Tasks View



3. Right-click in the **Target Tasks** view and select **New Task** from the context menu.

   The **Create New Target Task** wizard appears as shown in Figure 16.51.

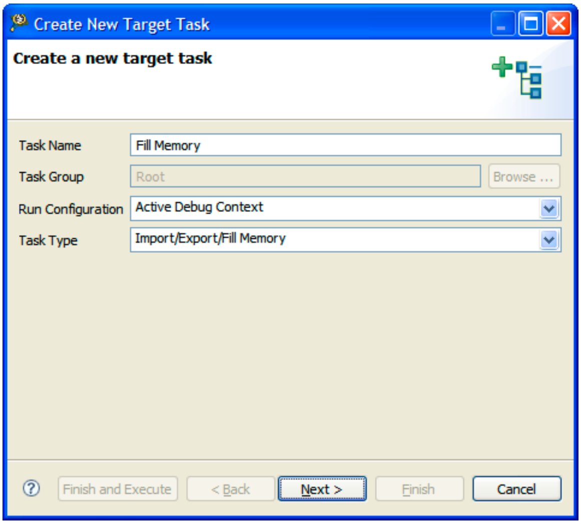
4. In the **Task Name** text box, enter a name for the new task. For example, *Fill Memory*.

5. Use the **Run Configuration** list box to specify the configuration that the task launches and uses to connect the target. For example, *Active Debug Context*.

NOTE If the task does not successfully launch the configuration that you specify, the **Execute** button of the **Target Tasks** view toolbar stays disabled.

6. From the **Task Type** list box, select **Import/Export/Fill Memory**.

Figure 16.51 Create New Target Task Wizard



Create New Target Task

Create a new target task

Task Name: Fill Memory
Task Group: Root
Run Configuration: Active Debug Context
Task Type: Import/Export/Fill Memory

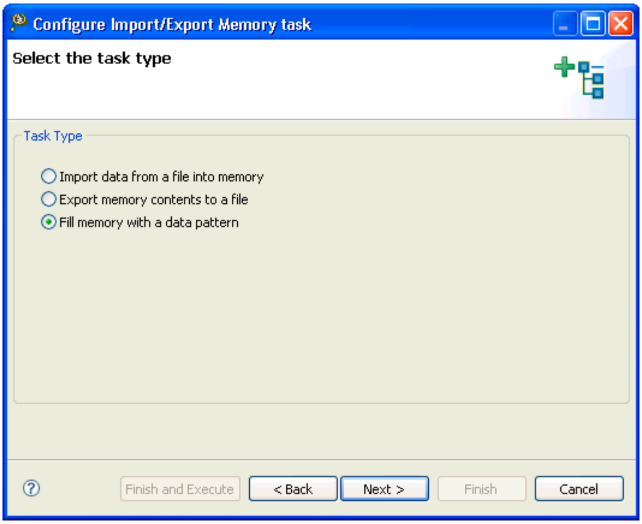
Browse ...
Finish and Execute
< Back
Next >
Finish
Cancel

7. Click **Next**.

   The **Select the task type** page (Figure 16.52) appears.

8. Select **Fill memory with a data pattern** task type option.

Figure 16.52 Create New Target Task Wizard — Select Task Type



Configure Import/Export Memory task

Select the task type

Task Type

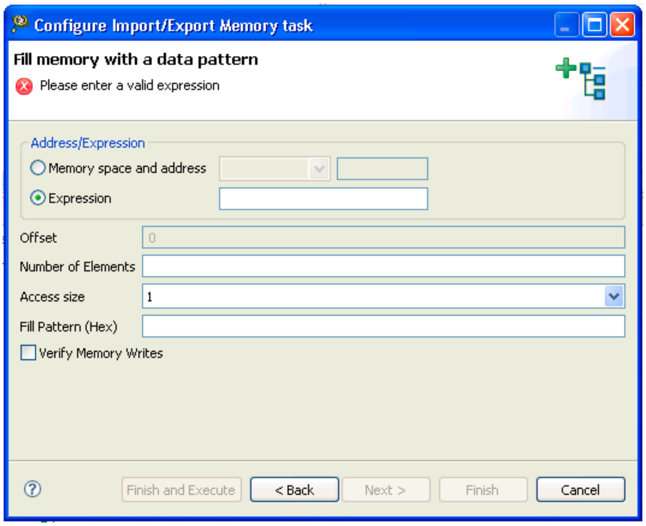
○ Import data from a file into memory
○ Export memory contents to a file
● Fill memory with a data pattern

Finish and Execute
< Back
Next >
Finish
Cancel

9. Click **Next**.

   The **Fill memory with a data pattern** page (Figure 16.53) appears. This page lets you fill a user specified memory range with a user specified data pattern.

Figure 16.53 Fill memory with a data pattern Page



Configure Import/Export Memory task
Fill memory with a data pattern
Please enter a valid expression
Address/Expression
○ Memory space and address
● Expression
Offset
0
Number of Elements
Access size
1
Fill Pattern (Hex)
☐ Verify Memory Writes
?
Finish and Execute
< Back
Next >
Finish
Cancel

10. Specify options as explained in Table 16.8.

NOTE CodeWarrior IDE validates information as you enter it. If there are errors, a message appears near the page title.

Table 16.8 Fill memory with a data pattern Page Options

Item	Description
Memory space and address	Enter the literal address and memory space on which the fill operation is performed. The Literal address field allows only decimal and hexadecimal values.
Expression	Enter the memory address or expression at which the fill operation starts.

Table 16.8 Fill memory with a data pattern Page Options

Item	Description
Offset	Enter a value to offset addresses contained in Motorola S-Record and Annotated Hex Text data formats. The field remains disabled for all other data formats.
Number of Elements	Enter the total number of elements to be modified.
Access Size	Denotes the number of addressable units of memory that the debugger accesses in modifying one data element. The default values shown are 1, 2, 4, and 8 units. When target information is available, this list shall be filtered to display the access sizes that are supported by the target.
Fill Pattern	Denotes the sequence of bytes, ordered from low to high memory, the wizard mirrors in the target. The field accept only hexadecimal values. If the width of the pattern exceeds the access size, the wizard displays an error message.
Verify Memory Writes	Check the option to verify success of each data write to the memory.

11. Click **Finish**.

    CodeWarrior IDE saves your changes, closes the **Configure Import/Export Memory task** wizard, and displays the newly created fill task in the **Tasks** list of the **Target Tasks** view.

NOTE Alternatively, click **Finish and Execute** to save your changes and execute the newly created fill task immediately.

Index

Numerics

16-Bit Analog-to-Digital Converter Module 815

A

ABI 23
About this Manual 13
Absolute Assembly 45, 69, 78
access size 1117, 1121, 1125
Action Type 1112
Actions 1076
actions 1070
Active Debug Context 1068, 1122
active debug session 1068
ADC 813
ADC Module Commands 814
ADCLR 813, 815
Add Checksum Action 1071
Add Device 1069
Add Device f 1069
Add Diagnostics Action 1071
Add Dump Flash Action 1071
Add Erase / Blank Check Action 1071
Add Program / Verify Action 1071, 1092
Add Protect/Unprotect Action 1071
ADDI 814, 815
ADDI Command 814
Additional Include Files (-AddInd) 242, 340
Address 1070
algorithm 1091
All options 211, 216, 232, 237, 310, 315, 330, 335, 405, 407, 414, 441, 444, 716
Allocate CONST objects in ROM 355
Alternate Load Address 589
Always Search User Paths (-nosyspath) 416, 473
Analog-to-Digital Converter Module 813
analysis 1063
ANSI startup code 48
Application 412
Apply 122
Architecture 714

B

Banked 49
bare board debugging 613
Bare Board Projects 29
basic 1111
Bigraph and trigraph support 246, 344
BlankCheck 1063
board 66, 74, 83, 91, 113, 648, 753, 804, 890, 956, 988, 1063, 1075
Breakpoints 629
breakpoints 629
bug 13
Build All 119
Build and Launch Configurations 29
Build Automatically 120
Build Project 119
Build Properties 191, 709
    HCS08 201, 299
build properties 709
Build Properties for HCS08 192, 710
Build Properties for RS08 192, 396
build properties panel
    HCS08 Compiler
        General 265
Building Project 29, 118
Burner 231, 329, 404

C

C 45, 46
C++ comments in ANSI-C 246, 344
C/C++ Build property 192, 194, 197, 710
C/C++ Options 47, 48
Change Device/Connection
    for ColdFire 955
    for HCS08 803

  - for PowerPC 1015
  - for RS08 889
- change direction 807
- Changing Build Properties 192, 710
- Changing P&E Connection Settings
  - for ColdFire V1 989
  - for ColdFire V2/3/4 957
  - for PowerPC 1017
  - for RS08 927
- Checksum 1063
- Chip View 804, 805
- Chip View Display After Change 809
- Chip View Display Before Change 808
- Clock Generation Module 817, 820
- Code Generation 233, 255, 266, 331, 333, 352, 364, 406
- CodeWarrior 15
- CodeWarrior Debugger 24
- CodeWarrior documentation 13
- CodeWarrior Download configuration 120, 650
- CodeWarrior IDE 13
  - compared to command-line tools 25
- CodeWarrior Project View 72, 81, 89, 96
- CodeWarrior Projects 804, 890, 956, 988
- CodeWarrior Projects View 72, 81, 89, 96
- CodeWarrior tools
  - using tools to debug programs 747
- ColdFire 1077, 1093
- ColdFire Build Options 49, 50, 52
- ColdFire CPU 398, 399
- ColdFire Debug 397
- ColdFire derivative 50, 51
- ColdFire Linker 403, 407, 457
- ColdFire Program 1093
- ColdFire uClinux Compiler
  - Miscellaneous 735
  - Warnings 733
- Command 211, 216, 232, 237, 310, 315, 330, 335, 405, 407, 414, 441, 444, 716
- Command line pattern 211, 217, 232, 237, 310, 315, 330, 335, 405, 408, 415, 441, 444, 716
- command-line tools and CodeWarrior IDE compared 25
- Common Connection Features 1063
- Compile only, So Nt Link (-c) 416, 473
- Compiler 413, 414
- Configure 1115
- Configure Import/Export Memory 1121, 1125
- Configure Import/Export Memory task 1115
- configuring LTIB 781
- Connections 34, 37, 39, 67, 85, 93, 644
- Connections Page 67, 69, 78, 93
- console connection, establishing 793
- Counter 819
- Create an MCU 65, 74, 83, 91, 113, 114, 753, 754
- Create an MCU Project Page 31
- Create Flash Programmer Task 1063
- Create New Target Task 1067, 1118, 1122
- Create New Target Task Wizard 1068
- Creating C++ Project 63
- Creating Project 62
- CWInstallDir 15
- Cycles Dialog Box with Cleared Counter 819

D

- Debug 122, 1113, 1118, 1122
- debug 13
- Debug Configurations Dialog Box
  - Debugger Page 122
- Debug Configurations dialog box 121
- Debug group 1064, 1066
- debug probe 792
  - USB Tap 792
- Debugger 122
- Debugger Page 122
- Debugger tab 121
- Debugging 400, 453
- debugging
  - bare board software 613
  - multiple cores 665
  - setting breakpoint types 629
  - setting watchpoint types 625
  - shared libraries 771–779
- Debugging Project 29, 120
- Debugging Projects 120
- default workspace 63, 72, 81, 89, 112, 645, 751
- Define preprocessor macros 239, 337
- DEMOS908QG8 1075

derivative 66, 74, 83, 91, 113, 648, 753, 804, 890, 956, 988
development process 13
Development Tools 22
development tools 13
Device Intialization 54
Device Package 805
Device/Connection Change 955, 987
  for HCS08 803
Devices 32, 66, 75, 77
devices 1070
diagnostic 1111
Diagnostics 1111
directories 1064
Disassembler 210, 212, 213, 309, 311
Do not use default libraries 718
Do not use MWCIncludes variable (-nostdinc) 416, 473
Do not use standard start files 718
Documentation 15
documentation 13
Download 1075, 1077
downloading 1074
DSP 23

E
Easy Debug 51, 53
EEPROM 821, 822
Emit #pragmas (-ppopt pragma) 442, 507
Emit file change (-ppopt break) 442, 507
Empty Declarations 421, 478
Enable C++ Support 51, 52
Enable Porting Support 51
EOL Character 413
Erase 1063
erase 1075
erasing on-chip memory 1074
establishing a console connection 793
ETSI 23
Execute 1075, 1076, 1077, 1095, 1103, 1118
execute 1065, 1100
Execute icon 1076, 1095, 1097, 1099, 1103
Expert Settings 211, 217, 232, 237, 310, 315, 330, 335, 405, 408, 415, 441, 444
Expert settings 716
Export 1113, 1117
Export data 1119
Export Data from Memory 1120
Export data from memory into a file 1119
Export Memory 1117
Expression 1116, 1120, 1124
Expression Has No Side Effect 420, 477
Extended Error Checking 419, 476
Extended Menu 806
Extra Commas 421, 478

F
features 13
File Type 1121
File type 1117
Fill 1113
Fill Memory 1122
Fill memory with a data pattern 1123
Fill Memory with Data Pattern 1121
Fill Pattern 1125
Finish 1112
fixes 13
flash memory 1063, 1074
Flash Programmer 1063, 1068
Flash Programmer Actions 1076, 1095, 1096, 1098
Flash Programmer Actions - Move Down 1071
Flash Programmer Actions - Move Up 1071
Flash Programmer Actions group 1065
Flash Programmer Actions -Remove Actions 1071
Flash Programmer target task 1066
Flash Programmer Task editor window 1065
Flash Programmer Window 1063
Flash Task 1077
Float is IEEE32, double is IEEE32 49
Float is IEEE64, double is IEEE64 51, 53
Frequency Display 818
Full Chip Simulation 803
Full Optimizations 51, 53
functional 1111

G

GCC toolchains 792
General 410, 717
Generate Binary Image 413
Generate Elf Symbol Table 413
Generate Link Map 412
Generate Link Map - Generate S-Record File 413
Generate Link Map - List Inused Objects 412
Generate Link Map -Always Keep Map 413
Generate Link Map -Show Transitive Closure 413
Generate Listing File 413
Generate Symbolic Info 400
Generate Warning Messages 413

H

hacking the kernel (debugging) 798
hardware 1063, 1066, 1111
Hardware Diagnosti 1112
Hardware Diagnostics 1111
Hardware Diagnostics Action 1112
hardware tools
    flash programmer 1063–1071
HCS08 1075
HCS08 Compiler 236, 335
HCS08 Compiler > General build properties panel 265
Help 15
Hidden virtual functions 420, 477

I

IDE 13
IDE project 1063
IICCLR 829, 832
IICDI 828, 830
IICDO 828
Illegal Pragmas 419, 476
Implicit Arithmentic Conversions 420, 477
Implicit Float to integer Conversions 420, 477
Implicit Integer to Float Conversions 420, 477
Implict Signed/Unsigned Conversion 420, 477
Import 1065, 1075, 1077, 1100, 1113
import 1064, 1075, 1094
Import Memory 1113
Import pre-defined task 1064
Import/Export 1115
Import/Export/Fill Memory 1118, 1122
In-Circuit Debugging 804
Include File Capitalization 421, 478
Include files only once 242, 340
incompatibilities 13
Inconsistent ‘class’ / ‘struct’ Usage 421, 478
Information 15
initialization file, JTAG 665
INPUT 833, 835, 836
Input 240, 283, 338, 380, 416, 436, 473
Input File 1117
INPUTS 832, 834, 835, 837
IRQ 833, 835

J

JTAG Chain
    multiple cores 665
JTAG initialization file 665

K

Keep comment (-ppopt comment) 443, 508
Keep whitespace (-ppopt nospace) 443
kernel debugging
    establishing a console connection 793
    kernel hacking option 798
    LTIB configuration 781
    setting up target hardware 792
kernel hacking 798

L

-L 720
-l 720
Language 244, 285, 342, 382
Language Page 44
LCF 24
Libraries 719, 720
Library Files ?(-l +file) 410, 462
Library search path 720
Linker 215, 314
Linker > Libraries 217, 316, 411, 413

Linux 709
Linux GCC toolchains 792
Linux Projects 30
Linux/uClinux 710
loat is IEEE32, double is IEEE64 49
Location 32, 641
Loop Speed 1113
LTIB configuration 781

M
Manipulating 1113
Max Bin Record 413
Max S-Record Length 413
Maximum Number of Errors 206, 305, 402, 456
Maximum Number of Warnings 206, 305, 402, 456
MC9S08QG8 1076
Memory 1113
memory 1063, 1069, 1074
Memory Access 1113
Memory into File 1120
Memory Model 210, 309
Memory read/write 1111
Memory Settings 1076
Memory space and address 1116, 1120, 1124
Memory Tests 1113
Memory tests 1111
Message Style 402, 456
Messages 204, 401
Microcontroller 1093
Minimal startup code 48
Missing 'return' Statement 420, 477
MMU View
    open 683
MMU Viewer 683
Module Options 811
Move Down 1071
Move Up 1071

N
New Bareboard Project 62, 63, 64, 65, 66, 67, 69, 70, 72, 74, 75, 76, 77, 78, 79, 81, 83, 84, 85, 87, 89, 90, 91, 92, 111, 113, 114, 750, 752, 753, 754
New Bareboard Project Wizard 29, 30
    Connections Page 67, 93
    Devices 77
    Devices Page 75
New Linux 709
New Linux/uClinux Application Project 62
New Target Task 1114, 1123
No Optimizations 51, 53
No Porting Support 51
No shared libraries 718
No startup or default libs 718
-nodefaulltlibs 718
None 54
-nostartfiles 718
-nostdlib 718
Number of Elements 1116, 1121, 1125

O
Offset 1116, 1121, 1125
Omit all symbol information 718
on-chip memory 1074
Open 1075, 1094, 1096, 1098
Other 1064, 1066
Other Flags 411, 434
Output 212, 266, 267, 290, 311, 354, 355, 364, 365, 387, 411
Output File 1121
overview 13

P
P&E Cyclone MAX Serial Connection 1030
P&E Cyclone MAX TCP-IP Connection 1033
P&E Cyclone MAX USB Connection 1031
P&E Full Chip Simulation 804
P&E Serial Cyclone Max 38
P&E TCP-IP Cyclone Max 38
P&E USB Cyclone Max 38
P&E USB PPCNexus Multilink 1028
PIC 588
Pointer/Integral Conversions 420, 477
Popup 1070
Ports Module Support 807
Possible Errors 419, 476
PPAGE 824
PPCNEXUS P&E Multilink/Cyclone Max 1016

Pre-defined 1064
pre-defined tasks 1065, 1100
Preprocessor 237, 242, 336, 340, 441
process 13
Processor Expert 54
Processor Family (-proc) 399
Program 1063
program 1063, 1075
Programming 1064
Programming Task 1063
Project name 31, 641
project settings
    static library 762
Propagate const and colatile qualifiers for structs 246, 344
PTA2 pin 808

R

RAD options 53
rapid 53, 1136
Rapid Application Development 53, 80, 88
Rapid Application Development Page 53
references 13
related 15
related documentation 13
Release Notes 13
release notes 13, 15
Relocatable Assembly 45, 69, 78
Remove Action 1071
Restoring Build Properties 194, 712
Revert 122
Run Configuration 1067, 1068, 1112, 1114, 1118
Run till a specific cycle 819
Run till Cycle 819

S

-s 718
Scope loop 1111
select 1075, 1094
Select Task Type 1115, 1123
Select the task type 1115
sequence 1066
setting up target hardware, kernel debugging 792
Settings. 192, 710
Show full path (-ppopt full) 442, 508
Show View 1064, 1066, 1113, 1117, 1122
Show View window 1064, 1066
Small 49
Sources 15
special 1066
specific device 1069
-static 718
static library project settings 762
steps 13
Stored Actions 1066
Strict ANSI 245, 343
Strip path information 359
System Path 417, 437, 474, 498

T

Target 1113
target board 1063
target board debugging 613
Target RAM 1070
Target Task 1113
target task 1066
Target Task Wizard 1119
Target Tasks 1064, 1066, 1075, 1095, 1096, 1098, 1122
target tasks 1066
Target Tasks View 1065, 1067, 1114
Target Tasks view 1064, 1067
Task 1064, 1075, 1076, 1077
Task Name 1067
Task Type 1068, 1112
test 1063
tests 1111
third-party 13
TLB page 680
TLB Table 683
toolbar 1067
toolchains, GCC 792
Treat All Warnings 419, 476
Tutorial 1074
Tutorial A
    Import and Execute HCS08 Download Task 1075
Tutorial B

    Import and Execute ColdFire Download Task 1077
Tutorial C
    Create Erase Memory Task for HCS08 1079, 1108
Tutorial D
    Create Erase Flash Memory Task for ColdFire 1084
Tutorial E
    Create Download Program Task for ColdFire 1090
tutorials 124, 1074

U

uClinux 709
uClinux Linker 715
Unused Arguments 420, 477
Unused Variables 420, 477
USB cable 1075
USB Tap 792
Use #include line (-ppopt line) 443
Use default location 31, 641
User Path (-i) 417, 436, 474, 498

V

value 807
Verify 1063
Verify Memory Writes 1117, 1125

W

Warnings 418
watchpoints 625
Workbench window 71, 80, 88, 95, 804, 890, 956, 988
Working with Projects 29
WorkSpace Launcher 63, 72, 81, 89, 111, 645, 751
write 13
writing 1074
writing to flash memory 1074

X

xml file 1065, 1100
XTAL 818, 820